# INTRODUÇÃO A BANCO DE DADOS

Osvaldo Kotaro Takai
Isabel Cristina Italiano
João Eduardo Ferreira
DCC-IME-USP – Fevereiro - 2005

# Índice

## Prefácio

Esta apostila foi escrita para apoiar a aprendizagem dos alunos nas disciplinas de introdução a Sistemas Banco de Dados do IME-USP. Seu conteúdo é uma pesquisa de vários autores, sendo em partes transcrições e traduções dos mesmos. Os capítulos de 1 a 9 foram baseados nas referências [1], [2], [3], [4], [5], [6] e [7]. O Capítulo 10 foi baseado nas demais referências que constam no final deste documento. O Apêndice A foi baseado em [5].

Esta apostila tem como objetivo ser uma primeira leitura para os alunos iniciantes no curso de banco de dados e tenta sempre mostrar os temas abordados de forma simples e clara, propiciando subsídios para aprofundar-se nos temas tratados utilizando outras bibliografias.

No intuito de ser didática, esta apostila está estruturada da seguinte forma:

O Capítulo 1 apresenta uma introdução aos modelos e arquiteturas de banco de dados.

O Capítulo 2 traz os conceitos básicos de sistemas de banco de dados, necessários à compreensão do restante deste material.

No Capítulo 3, as arquiteturas de banco de dados são apresentadas.

O Capítulo 4 trata de modelagem de banco de dados usando o paradigma entidade relacionamento.

O Capítulo 5 aborda o modelo relacional.

O Capítulo 6 ilustra como efetuar o mapeamento de um diagrama entidade relacionamento para o modelo relacional, de forma intuitiva.

O Capítulo 7 traz uma discussão e exemplos de linguagens de consultas formais, quais sejam: Álgebra relacional, Cálculo relacional de Tuplas e Cálculo Relacional de Domínio.

No Capítulo 8, a linguagem SQL – linguagem de consulta comercial mais difundida para o modelo relacional – é introduzida, por meio de sua teoria e exemplos práticos.

O Capítulo 9 trata de projeto de banco de dados, normalização e dependências funcionais entre os dados.

No Capítulo 10 uma visão geral sobre Data warehouse é fornecida ao leitor. Finalmente, o Apêndice A traz exemplos de consultas nas linguagens de consultas vistas anteriormente.

Os autores agradecem imensamente aos alunos: Bianka M.M.T. Gonçalves, Clodis Boscarioli, Rodolpho Iemini Atoji e Fernando Henrique Ferraz P. da Rosa, pelas valorosas correções, edições e sugestões do texto em questão.

**Base de Dados**

# 1 Introdução

O primeiro Sistema Gerenciador de Banco de Dados (SGBD) comercial surgiu no final de 1960 com base nos primitivos sistemas de arquivos disponíveis na época, os quais não controlavam o acesso concorrente por vários usuários ou processos. Os SGBDs evoluíram desses sistemas de arquivos de armazenamento em disco, criando novas estruturas de dados com o objetivo de armazenar informações. Com o tempo, os SGBD's passaram a utilizar diferentes formas de representação, ou modelos de dados, para descrever a estrutura das informações contidas em seus bancos de dados. Atualmente, os seguintes modelos de dados são normalmente utilizados pelos SGBD's: modelo hierárquico, modelo em redes, modelo relacional (amplamente usado) e o modelo orientado a objetos.

## 1.1 Modelos de Dados

### 1.1.1 Modelo Hierárquico

O modelo hierárquico foi o primeiro a ser reconhecido como um modelo de dados. Seu desenvolvimento somente foi possível devido à consolidação dos discos de armazenamento endereçáveis, pois esses discos possibilitaram a exploração de sua estrutura de endereçamento físico para viabilizar a representação hierárquica das informações. Nesse modelo de dados, os dados são estruturados em hierarquias ou árvores. Os nós das hierarquias contêm ocorrências de registros, onde cada registro é uma coleção de campos (atributos), cada um contendo apenas uma informação. O registro da hierarquia que precede a outros é o registro-pai, os outros são chamados de registros-filhos.

Uma ligação é uma associação entre dois registros. O relacionamento entre um registro-pai e vários registros-filhos possui cardinalidade 1:N. Os dados organizados segundo este modelo podem ser acessados segundo uma seqüência hierárquica com uma navegação do topo para as folhas e da esquerda para a direita. Um registro pode estar associado a vários registros diferentes, desde que seja replicado. A replicação possui duas grandes desvantagens: pode causar inconsistência de dados quando houver atualização e o desperdício de espaço é inevitável. O sistema comercial mais divulgado no modelo hierárquico foi o *Information Management System da IBM Corp(IMS*). Grande parte das restrições e consistências de dados estava contida dentro dos programas escritos para as aplicações. Era necessário escrever programas na ordem para acessar o banco de dados. Um diagrama de estrutura de árvore descreve o esquema de um banco de dados hierárquico. Tal diagrama consiste em dois componentes básicos: Caixas, as quais correspondem aos tipos de registros e Linhas, que correspondem às ligações entre os tipos de registros. Como exemplo do modelo hierárquico, considere a Figura 1.1 abaixo.

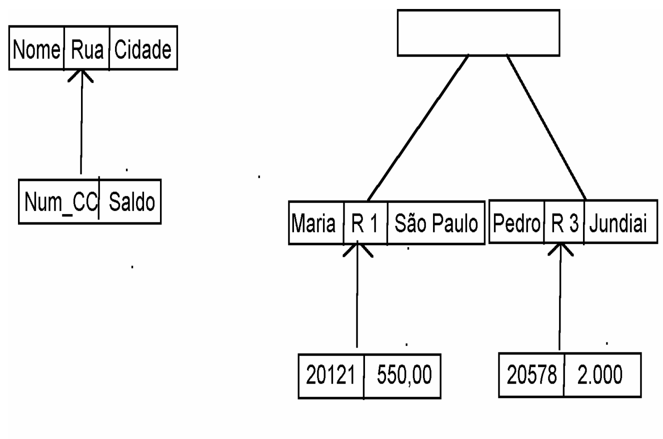



**Figura 1.1 - Diagrama de estrutura de árvore Cliente - Conta Corrente**

### 1.1.2 Modelo em Rede

O modelo em redes surgiu como uma extensão ao modelo hierárquico, eliminando o conceito de hierarquia e permitindo que um mesmo registro estivesse envolvido em várias associações. No modelo em rede, os registros são organizados em grafos onde aparece um único tipo de associação (*set*) que define uma relação 1:N entre 2 tipos de registros: proprietário e membro. Desta maneira, dados dois relacionamentos 1:N entre os registros A e D e entre os registros C e D é possível construir um relacionamento M:N entre A e D. O gerenciador *Data Base Task Group (DBTG) da CODASYL (Committee on Data Systems and Languages)* estabeleceu uma norma para este modelo de banco de dados, com linguagem própria para definição e manipulação de dados. Os dados tinham uma forma limitada de independência física. A única garantia era que o sistema deveria recuperar os dados para as aplicações como se eles estivessem armazenados na maneira indicada nos esquemas. Os geradores de relatórios da CODASYL também definiram sintaxes para dois aspectos chaves dos sistemas gerenciadores de dados: concorrência e segurança. O mecanismo de segurança fornecia uma facilidade na qual parte do banco de dados (ou área) pudesse ser bloqueada para prevenir acessos simultâneos, quando necessário. A sintaxe da segurança permitia que uma senha fosse associada a cada objeto descrito no esquema. Ao contrário do Modelo Hierárquico, em que qualquer acesso aos dados passa pela raiz, o modelo em rede possibilita acesso a qualquer nó da rede sem passar pela raiz. No Modelo em Rede o sistema comercial mais divulgado é o CA-IDMS da *Computer Associates*. O diagrama para representar os conceitos do modelo em redes consiste em dois componentes básicos: Caixas, que correspondem aos registros e Linhas, que correspondem às associações. A Figura 1.2 ilustra um exemplo de diagrama do modelo em rede.

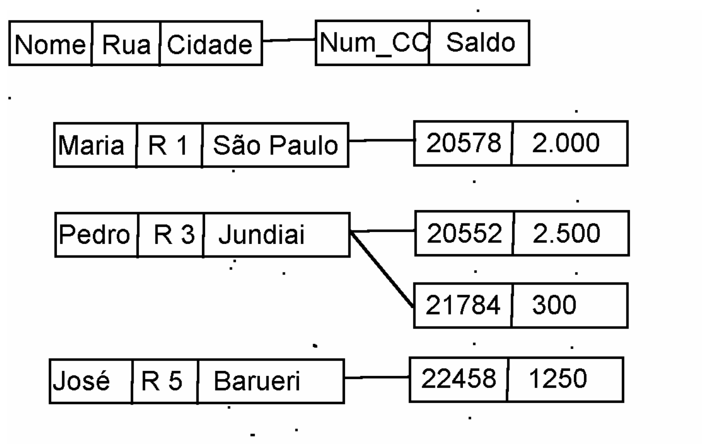



**Figura 1.2 - Diagrama de estrutura de dados Cliente - Conta Corrente**

Estes dois modelos: Hierárquico e Rede são Orientados a Registros, isto é, qualquer acesso à base de dados – inserção, consulta, alteração ou remoção – é feito em um registro de cada vez.

### 1.1.3 Modelo Relacional

O modelo relacional apareceu devido às seguintes necessidades: aumentar a independência de dados nos sistemas gerenciadores de banco de dados; prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados; permitir processamento *ad hoc*[1]. O modelo relacional, tendo por base a teoria dos conjuntos e álgebra relacional, foi resultado de um estudo teórico realizado por CODD[1][2]. O Modelo relacional revelou-se ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados. A estrutura fundamental do modelo relacional é a relação (tabela). Uma relação é constituída por um ou mais atributos (campos) que traduzem o tipo de dados a armazenar. Cada instância do esquema (linha) é chamada de tupla (registro). O modelo relacional não tem caminhos pré-definidos para se fazer acesso aos dados como nos modelos que o precederam. O modelo relacional implementa estruturas de dados organizadas em relações. Porém, para trabalhar com essas tabelas, algumas restrições precisaram ser impostas para evitar aspectos indesejáveis, como: Repetição de informação, incapacidade de representar parte da informação e perda de informação. Essas restrições são: integridade referencial, chaves e integridade de junções de relações. A Figura 1.3, abaixo, traz exemplos de tabelas sob o modelo relacional.

| Cod_Cliente | Nome | Rua | Cidade |
|---|---|---|---|
| 1 | Pedro | A | São Paulo |
| 2 | Maria | B | Jundiai |
| | | | |

| Num_CC | Saldo |
|---|---|
| 20121 | 1200 |
| 21582 | 1320 |
| 21352 | 652 |

| Cod_Cliente | Num_CC |
|---|---|
| 1 | 20121 |
| 2 | 21582 |
| 2 | 21352 |

**Figura 1.3 Tabelas do modelo relacional Cliente - Conta Corrente**

### 1.1.4 Modelo Orientado a Objetos

Os bancos de dados orientados a objeto começaram a se tornar comercialmente viáveis em meados de 1980. A motivação para seu surgimento está em função dos limites de armazenamento e representação semântica impostas no modelo relacional. Alguns exemplos são os sistemas de informações geográficas (SIG), os sistemas CAD e CAM, que são mais facilmente construídos usando tipos complexos de dados. A habilidade para criar os tipos de dados necessários é uma característica das linguagens de programação orientadas a objetos. Contudo, estes sistemas necessitam guardar representações das estruturas de dados que utilizam no armazenamento permanente. A estrutura padrão para os bancos de dados orientados a objetos foi feita pelo Object Database Management Group (ODMG). Esse grupo é

[1] Processamento dedicado, exclusivo.
[2] Codd era investigador da IBM. O modelo foi apresentado num artigo publicado em 1970, mas só nos anos 80 o modelo foi implementado.

formado por representantes dos principais fabricantes de banco de dados orientados a objeto disponíveis comercialmente. Membros do grupo têm o compromisso de incorporar o padrão em seus produtos. O termo Modelo Orientado a Objetos é usado para documentar o padrão que contém a descrição geral das facilidades de um conjunto de linguagens de programação orientadas a objetos e a biblioteca de classes que pode formar a base para o Sistema de Banco de Dados. Quando os bancos de dados orientados a objetos foram introduzidos, algumas das falhas perceptíveis do modelo relacional pareceram ter sido solucionadas com esta tecnologia e acreditava-se que tais bancos de dados ganhariam grande parcela do mercado. Hoje, porém, acredita-se que os Bancos de Dados Orientados a Objetos serão usados em aplicações especializadas, enquanto os sistemas relacionais continuarão a sustentar os negócios tradicionais, onde as estruturas de dados baseadas em relações são suficientes. O diagrama de classes UML serve geralmente como o esquema para o modelo de dados orientado a objetos. Observe o exemplo da Figura 1.4, e compare as diferenças com o modelo anterior.

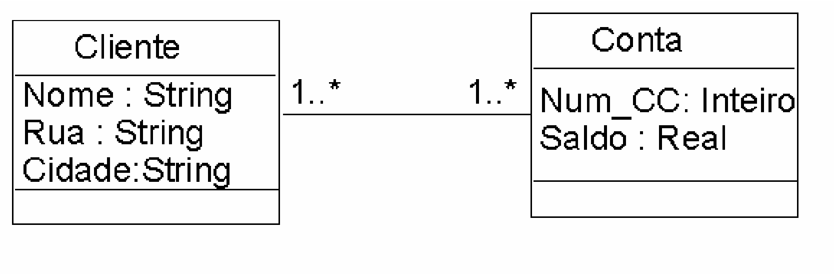



**Figura 1.4 - Diagrama UML Cliente - Conta Corrente**

### 1.1.5 Sistemas Objeto-Relacionais

A área de atuação dos sistemas Objeto-Relacional tenta suprir a dificuldade dos sistemas relacionais convencionais, que é o de representar e manipular dados complexos, visando ser mais representativos em semântica e construções de modelagens. A solução proposta é a adição de facilidades para manusear tais dados utilizando-se das facilidades SQL (*Structured Query Language*) existentes. Para isso, foi necessário adicionar: extensões dos tipos básicos no contexto SQL; representações para objetos complexos no contexto SQL; herança no contexto SQL e sistema para produção de regras.

## 1.2 Arquiteturas de Banco Dados

### 1.2.1 Introdução

Atualmente, devem-se considerar alguns aspectos relevantes para atingir a eficiência e a eficácia dos sistemas informatizados desenvolvidos, a fim de atender seus usuários nos mais variados domínios de aplicação: automação de escritórios, sistemas de apoio a decisões, controle de reserva de recursos, controle e planejamento de produção, alocação e estoque de recursos, entre outros. Tais aspectos são:

a) Os projetos Lógico e Funcional do Banco de Dados devem ser capazes de prever o volume de informações armazenadas a curto, médio e longo prazo. Os projetos devem ter uma grande capacidade de adaptação para os três casos mencionados;

b) Deve-se ter generalidade e alto grau de abstração de dados, possibilitando confiabilidade e eficiência no armazenamento dos dados e permitindo a utilização de diferentes tipos de gerenciadores de dados através de linguagens de consultas padronizadas;

c) Projeto de uma interface ágil e com uma "rampa ascendente" para propiciar aprendizado suave ao usuário, no intuito de minimizar o esforço cognitvo;

d) Implementação de um projeto de interface compatível com múltiplas plataformas (UNIX, Windows NT, Windows Workgroup, etc);

e) Independência de Implementação da Interface em relação aos SGBDs que darão condições às operações de armazenamento de informações (ORACLE, SYSBASE, INFORMIX, PADRÃO XBASE, etc).

f) Conversão e mapeamento da diferença semântica entre os paradigmas utilizados no desenvolvimento de interfaces (Imperativo (ou procedural), Orientado a Objeto, Orientado a evento), servidores de dados (Relacional) e programação dos aplicativos (Imperativo, Orientado a Objetos).

### 1.2.2 Arquiteturas

As primeiras arquiteturas usavam *mainframes* para executar o processamento principal e de todas as funções do sistema, incluindo os programas aplicativos, programas de interface com o usuário, bem como a funcionalidade dos SGBDs. Esta é a razão pela qual a maioria dos usuários fazia acesso aos sistemas via terminais que não possuíam poder de processamento, apenas a capacidade de visualização. Todos os processamentos eram feitos remotamente, apenas as informações a serem visualizadas e os controles eram enviados do *mainframe* para os terminais de visualização, conectados a ele por redes de comunicação. Como os preços do hardware foram decrescendo, muitos usuários trocaram seus terminais por computadores pessoais (PC) e estações de trabalho. No começo os SGBDs usavam esses computadores da mesma maneira que usavam os terminais, ou seja, o SGBD era centralizado e toda sua funcionalidade, execução de programas aplicativos e processamento da interface do usuário eram executados em apenas uma máquina. Gradualmente, os SGBDs começaram a explorar a disponibilidade do poder de processamento no lado do usuário, o que levou à arquitetura cliente-servidor.

A arquitetura cliente-servidor foi desenvolvida para dividir ambientes de computação onde um grande número de PCs, estações de trabalho, servidores de arquivos, impressoras, servidores de banco de dados e outros equipamentos são conectados juntos por uma rede. A idéia é definir servidores especializados, tais como servidor de arquivos, que mantém os arquivos de máquinas clientes, ou servidores de impressão que podem estar conectados a várias impressoras; assim, quando se desejar imprimir algo, todas as requisições de impressão são enviadas a este servidor. As máquinas clientes disponibilizam para o usuário as interfaces apropriadas para utilizar esses servidores, bem como poder de processamento para executar aplicações locais. Esta arquitetura se tornou muito popular por algumas razões. Primeiro, a facilidade de implementação dada a clara separação das funcionalidades e dos servidores. Segundo, um servidor é inteligentemente utilizado porque as tarefas mais simples são delegadas às máquinas clientes mais baratas. Terceiro, o usuário pode executar uma interface gráfica que lhe é familiar, ao invés de usar a interface do servidor. Desta maneira, a arquitetura cliente-servidor foi incorporada aos SGBDs comerciais. Diferentes técnicas foram propostas para se implementar essa arquitetura, sendo que a mais adotada pelos Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDRs) comerciais é a inclusão da funcionalidade de um SGBD centralizado no lado do servidor. As consultas e a funcionalidade transacional permanecem no servidor, sendo que este é chamado de servidor de consulta ou servidor de transação. É assim que um servidor SQL é fornecido aos clientes. Cada cliente tem que formular suas consultas SQL, prover a interface do usuário e as funções de interface usando uma linguagem de programação. O cliente pode também se referir a um dicionário de dados o qual inclui informações sobre a distribuição dos dados em vários servidores SQL, bem como os módulos para a decomposição de uma consulta global em um número de consultas locais que podem ser executadas em vários sítios. Comumente o servidor SQL também é chamado de *back-end machine* e o cliente de *front-end machine*. Como SQL provê uma linguagem padrão para o SGBDRs, esta criou o ponto de divisão lógica entre o cliente e o servidor.

Atualmente, existem várias tendências para arquitetura de Banco de Dados, nas mais diversas direções.

### 1.2.3 Resumo das arquiteturas de SGBDs

- Plataformas centralizadas. Na arquitetura centralizada, existe um computador com grande capacidade de processamento, o qual é o hospedeiro do SGBD e emuladores para os vários aplicativos. Esta arquitetura tem como principal vantagem a de permitir que muitos usuários manipulem grande volume de dados. Sua principal desvantagem está no seu alto custo, pois exige ambiente especial para *mainframes* e soluções centralizadas.

- Sistemas de Computador Pessoal - PC. Os computadores pessoais trabalham em sistema *stand-alone*, ou seja, fazem seus processamentos sozinhos. No começo esse processamento era bastante limitado, porém, com a evolução do hardware, tem-se hoje PCs com grande capacidade de processamento. Eles utilizam o padrão Xbase e quando se trata de SGBDs, funcionam como hospedeiros e terminais. Desta maneira, possuem um único aplicativo a ser executado na máquina. A principal vantagem desta arquitetura é a simplicidade.

- Banco de Dados Cliente-Servidor. Na arquitetura Cliente-Servidor, o cliente (*front_end*) executa as tarefas do aplicativo, ou seja, fornece a interface do usuário (tela, e processamento de entrada e saída). O servidor (*back_end*) executa as consultas no DBMS e retorna os resultados ao cliente. Apesar de ser uma arquitetura bastante popular, são necessárias soluções sofisticadas de software que possibilitem: o tratamento de transações, as confirmações de transações (*commits*), desfazer transações (*rollbacks*), linguagens de consultas (*stored procedures*) e gatilhos (*triggers*). A principal vantagem desta arquitetura é a divisão do processamento entre dois sistemas, o que reduz o tráfego de dados na rede.

- Banco de Dados Distribuídos (N camadas). Nesta arquitetura, a informação está distribuída em diversos servidores. Como exemplo, observe a Figura 1.5. Cada servidor atua como no sistema cliente-servidor, porém as consultas oriundas dos aplicativos são feitas para qualquer servidor indistintamente. Caso a informação solicitada seja mantida por outro servidor ou servidores, o sistema encarrega-se de obter a informação necessária, de maneira transparente para o aplicativo, que passa a atuar consultando a rede, independente de conhecer seus servidores. Exemplos típicos são as bases de dados corporativas, em que o volume de informação é muito grande e, por isso, deve ser distribuído em diversos servidores. Porém, não é dependente de aspectos lógicos de carga de acesso aos dados, ou base de dados fracamente acopladas, em que uma informação solicitada vai sendo coletada numa propagação da consulta numa cadeia de servidores. A característica básica é a existência de diversos programas aplicativos consultando a rede para acessar os dados necessários, porém, sem o conhecimento explícito de quais servidores dispõem desses dados.

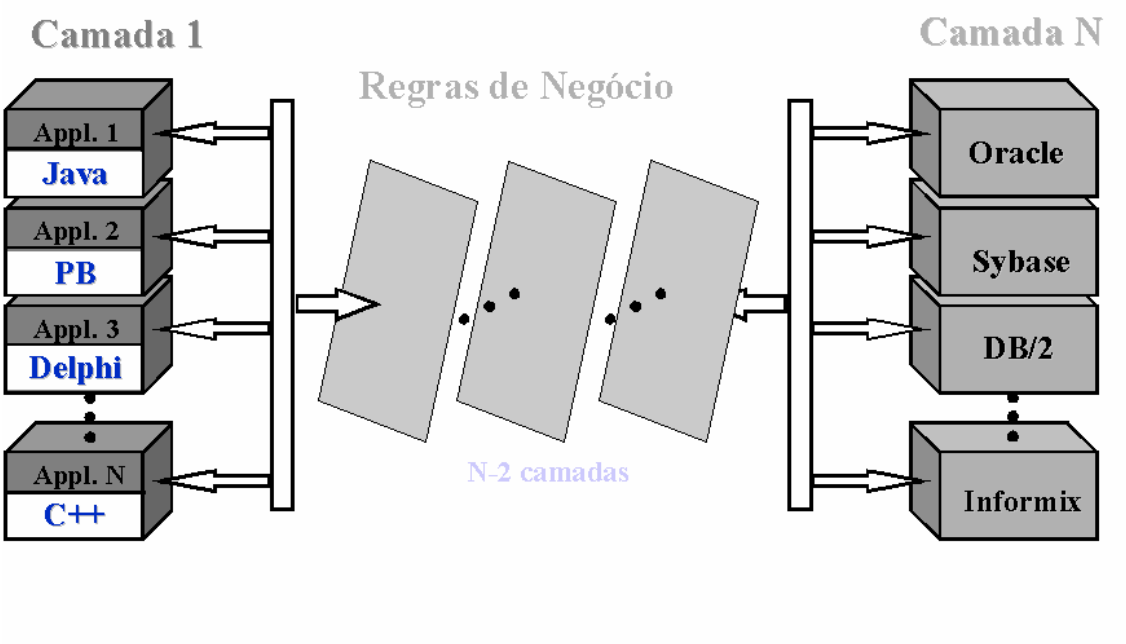



**Figura 1.5 - Arquitetura Distribuída N camadas**

## 1.3 Ambiente de Implementação Cliente-Servidor

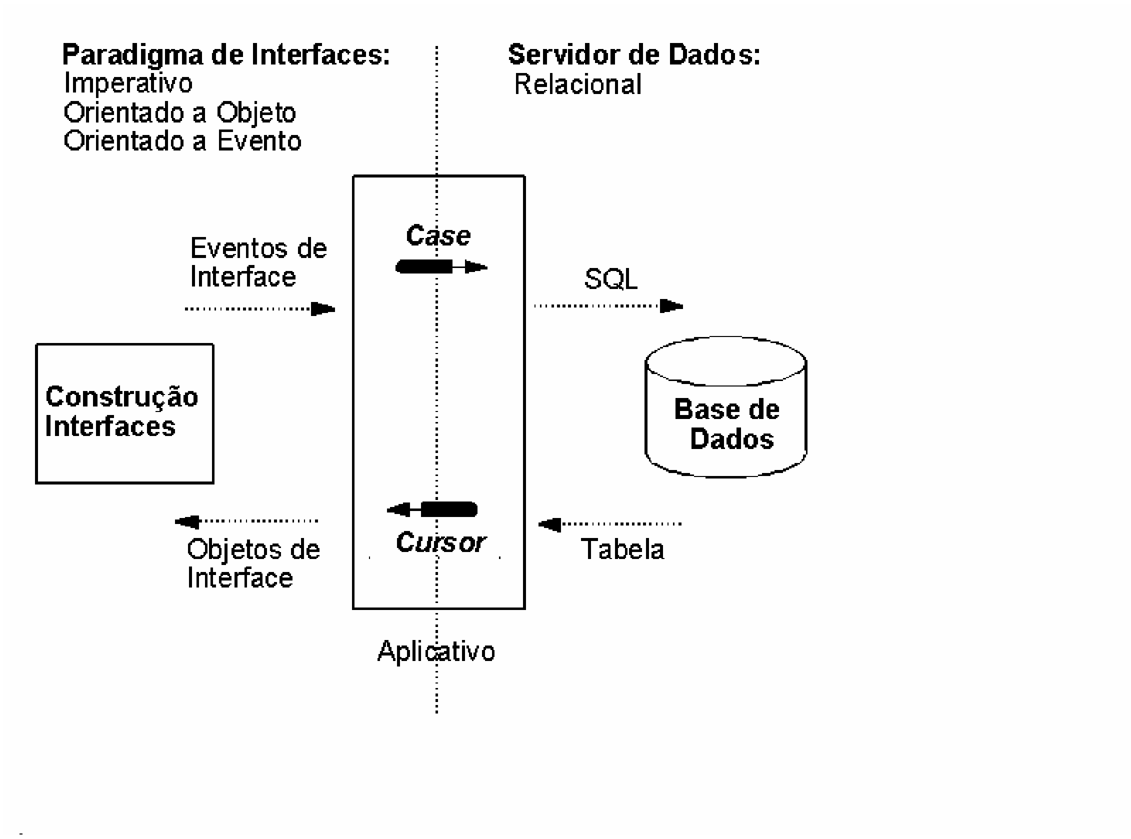


**Figura 1.6 - Ambiente genérico para desenvolvimento de aplicativos**

A Figura 1.6 ilustra um ambiente genérico de desenvolvimento de aplicativos. Nesta Figura, a diferença (gap semântico) entre os paradigmas utilizados para: a construção de interfaces, o armazenamento de informações, e a programação dos aplicativos são detalhadas para ressaltar a importância de estruturas "Case" e "Cursores". As estruturas "Case" são utilizadas para converter as alterações e solicitações ocorridas na interface do aplicativo em uma linguagem que seja capaz de ser processada pelos servidores de dados. A construção da linguagem é feita através da composição de cadeias de caracteres usualmente utilizando o padrão SQL utilizado nos servidores de dados relacionais. Quando um acesso ao SGBD é requerido, o programa estabelece uma conexão com o SGBD que está instalado no servidor. Uma vez que a conexão é criada, o programa cliente pode se comunicar com o SGBD. Um padrão chamado de Conectividade Base de Dados Aberta (*Open DataBase Connectivity - ODBC*) provê uma Interface para Programação de Aplicações (*API*) que permite que os programas no lado cliente possam chamar o SGBD, desde que as máquinas clientes como o servidor tenham o necessário software instalado. Muitos vendedores de SGBDs disponibilizam *drivers* específicos para seus sistemas. Desta maneira, um programa cliente pode se conectar a diversos SGBDRs e enviar requisições de consultas e transações usando API, que são processados nos servidores. Após o processamento de uma chamada de função (levando uma cadeia de caracteres ou programas armazenados), o resultado é fornecido pelo servidor de dados através de tabelas em memória. Os resultados das consultas são enviados para o programa cliente, que pode processá-lo ou visualizá-lo conforme a necessidade. O conjunto resposta para uma consulta pode ser uma tabela com zero, uma ou múltiplas tuplas, dependendo de quantas linhas foram encontradas com o critério de busca. Quando uma consulta retorna múltiplas linhas, é necessário declarar um "CURSOR" para processá-las. Um cursor é similar a uma variável de arquivo ou um ponteiro de arquivo, que aponta para uma única linha (tupla) do resultado da consulta. Em SQL os cursores são controlados por três comandos: OPEN, FETCH, CLOSE. O cursor é iniciado com o comando OPEN, que executa a consulta, devolve o conjunto resultante de linhas e coloca o cursor para a posição anterior à primeira linha do resultado da consulta. O comando FETCH, quando executado pela primeira vez, devolve a primeira linha nas variáveis do programa e coloca o cursor para apontar para aquela linha. Subseqüentes execuções do comando FETCH avançam o cursor para a próxima linha no conjunto resultante e retornam a linha nas variáveis do programa. Quando a última linha é processada, o cursor é desbloqueado com o comando CLOSE. Os cursores existem principalmente para que linguagens de programação que não permitem abstração para conjunto de registros, como C, possam receber as linhas da resposta de uma consulta SQL uma de cada vez. Com a utilização de "CURSORES", apresentam-se esses dados como resultados das consulta, através de itens que representam os elementos de interface com o usuário, atendendo os preceitos impostos pelos diferentes paradigmas possivelmente envolvidos. Com isso os resultados são mostrados utilizando o objeto padrão da interface, disponíveis nas ferramentas de construção de interfaces. Dessa forma, o ciclo de busca de informação nos mais variados servidores tem início e fim na interface com o usuário.

É de fundamental importância que se construam aplicativos cujos projetos de interface sejam "ortogonais" aos projetos de implementação de acesso aos servidores de dados. Na implementação de sistemas de informação, deve-se utilizar uma arquitetura de base de dados relacional que seja independente de um determinado repositório de dados (gerenciadores Access, Oracle, Sybase, Informix, etc).

A Figura 1.7 ilustra a utilização de conversores genéricos tanto para interfaces como para os servidores de dados. Estes conversores são construídos para padronizar o controle de compartilhamento de dados, independente da ferramenta de interface ou do servidor de dados. Em situações práticas esses conversores são denominados comumente de *drivers*.

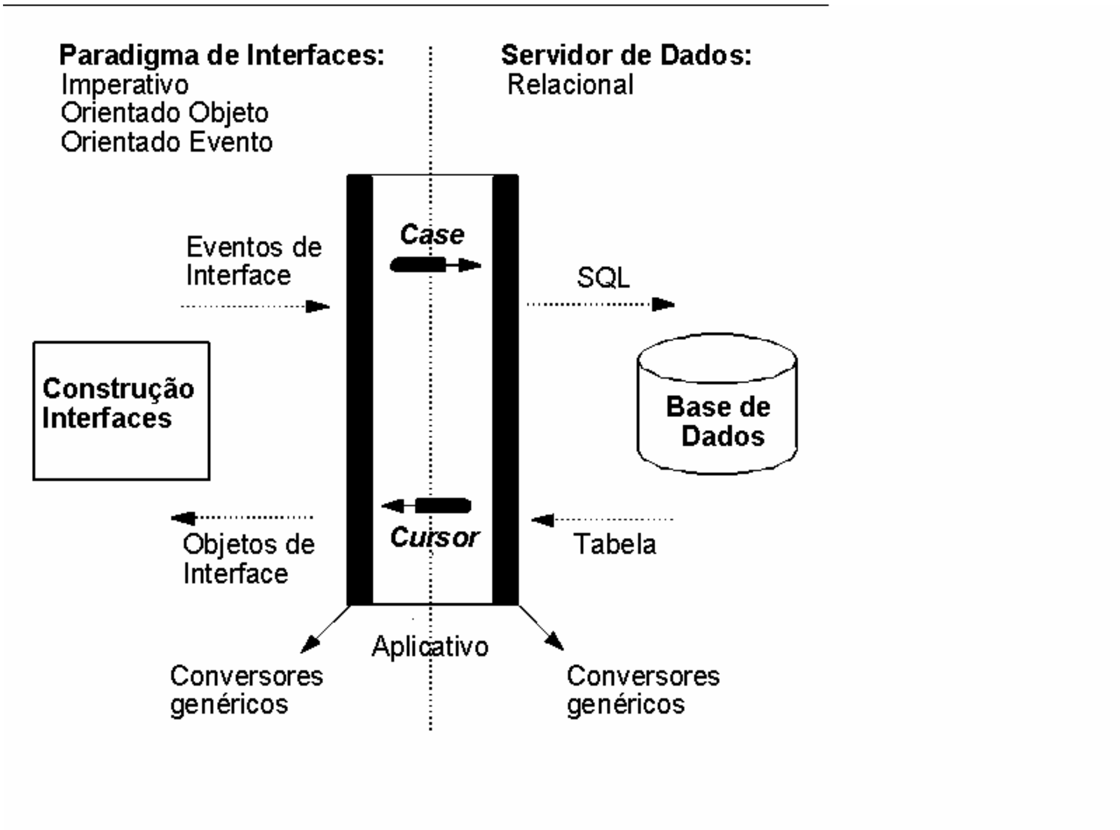



**Figura 1.7 - Conversor genérico para interface e servidores de dados**

# 2 Definição Geral

Base de Dados: Coleção de dados relacionados;

Dados: Valor de um campo armazenado, matéria-prima para obtenção de informação;

Informação: Dados compilados e processados de acordo com solicitação de consultas e análises

## 2.1 Propriedades:

- Uma base de dados é uma coleção de dados logicamente relacionados, com algum significado. Associações aleatórias de dados não podem ser chamadas de base de dados;
- Uma base de dados é projetada, construída e preenchida (instanciada) com dados para um propósito específico. Ela tem um grupo de usuários e algumas aplicações pré-concebidas para atendê-los;
- Uma base de dados representa algum aspecto do mundo real, algumas vezes chamado de "mini-mundo". Mudanças no mini-mundo provocam mudanças na base de dados.

Uma base de dados tem alguma fonte de dados, algum grau de interação com eventos do mundo real e uma audiência que está ativamente interessada no seu conteúdo.

Um Sistema Gerenciador de Base de Dados (SGBD) é uma coleção de programas que permitem aos usuários criarem e manipularem uma base de dados. Um SGBD é, assim, um sistema de software de propósito geral que facilita o processo de definir, construir e manipular bases de dados de diversas aplicações.

Definir uma base de dados envolve a especificação de tipos de dados a serem armazenados na base de dados.

Construir uma base de dados é o processo de armazenar os dados em algum meio que seja controlado pelo SGBD.

Manipular uma base de dados indica a utilização de funções como a de consulta, para recuperar dados específicos, modificação da base de dados para refletir mudanças no mini-mundo (inserções, atualizações e remoções), e geração de relatórios.

A base de dados e o software de gerenciamento da base de dados compõem o chamado Sistema de Base de Dados. A Figura 2.1 apresenta um esquema genérico de um Sistema de Banco de Dados em sua interação com seus usuários.

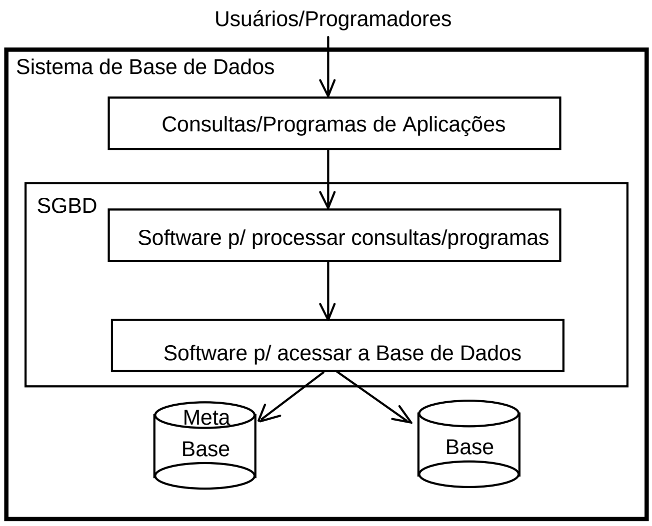



**Figura 2.1 – Sistema de Banco de Dados**

## 2.2 Características da Abordagem de Base de Dados x Processamento Tradicional de Arquivos

A tabela abaixo traz as principais características que diferem um sistema desenvolvido na perspectiva de banco de dados versus um desenvolvimento pelo tradicional gerenciamento de arquivos.

| Processamento tradicional de Arquivos | Base de Dados | Vantagens da base de Dados |
|---|---|---|
| Definição dos dados é parte do código de programas de aplicação | Meta Dados | eliminação de redundâncias |
| Dependência entre aplicação específica e dados | Capaz de permitir diversas aplicações | eliminação de redundâncias |
| | independência entre dados e programas | facilidade de manutenção |
| Representação de dados ao nível físico | Representação conceitual através de dados e programas | facilidade de manutenção |
| Cada visão é implementada por módulos específicos | Permite múltiplas visões | facilidade de consultas |

## 2.3 Capacidades do SGBD

- Controle de Redundância: no processamento tradicional de arquivos, muitos grupos de usuários mantêm seus próprios arquivos para manipular suas aplicações de processamento, que pode provocar o armazenamento de informações redundantes;
  Problemas:
  ⇒ Duplicação de esforços;
  ⇒ Desperdício de espaço;
  ⇒ Inconsistência: alteração em alguns arquivos e em outros não, ou em todos os arquivos, porém, de maneira independente;
- Compartilhamento de Dados: SGBD's multiusuários devem fornecer controle de concorrência para assegurar que atualizações simultâneas resultem em modificações corretas. Um outro mecanismo que permite a noção de compartilhamento de dados em um SGBD multiusuário é a facilidade de definir visões de usuário, que é usada para especificar a porção da base de dados que é de interesse para um grupo particular de usuários;
- Restrições de Acesso Multiusuário: quando múltiplos usuários compartilham uma base de dados, é comum que alguns usuários não autorizados não tenham acesso a todas as informações da base de dados. Por exemplo, os dados financeiros são freqüentemente considerados confidenciais e, desse modo, somente pessoas autorizadas devem ter acesso. Além disso, pode ser permitido a alguns usuários, apenas a recuperação dos dados. Já, para outros, são permitidas a recuperação e a modificação. Assim, o tipo de operação de acesso - recuperação ou modificação - pode também ser controlado. Tipicamente, usuários e grupos de usuários recebem uma conta protegida por palavras-chaves, que é usada para se obter acesso à base de dados, o que significa dizer que contas diferentes possuem restrições de acesso diferentes. Um SGBD deve fornecer um subsistema de autorização e segurança, que é usado pelo DBA para criar contas e especificar restrições nas contas. O SGBD deve então obrigar estas restrições automaticamente. Note que um controle similar pode ser aplicado ao software do SGBD;
- Fornecimento de Múltiplas Interfaces: devido aos vários tipos de usuários, com variados níveis de conhecimento técnico, um SGBD deve fornecer uma variedade de interfaces atendê-los. Os tipos de interfaces incluem linguagens de consulta para usuários ocasionais, interfaces de linguagem de programação para programadores de aplicações, formulários e interfaces dirigidas por menus para usuários comuns;
- Representação de Relacionamento Complexo entre Dados: uma base de dados pode possuir uma variedade de dados que estão inter-relacionados de muitas maneiras. Um SGBD deve ter a capacidade de representar uma variedade de relacionamentos complexos entre dados, bem como recuperar e modificar dados relacionados de maneira fácil e eficiente;
- Reforçar Restrições de Integridade: muitas aplicações de base de dados terão certas restrições de integridade de dados. A forma mais elementar de restrição de integridade é a

especificação do tipo de dado de cada item. Existem tipos de restrições mais complexas. Um tipo de restrição que ocorre freqüentemente é a especificação de que um registro de um arquivo deve estar relacionado a registros de outros arquivos. Um outro tipo de restrição especifica a unicidade sobre itens de dados. Estas restrições são derivadas da semântica dos dados e do mini-mundo que eles representam. Algumas restrições podem ser especificadas ao SGBD e automaticamente executadas. Outras restrições podem ser verificadas pelos programas de atualização ou no tempo da entrada de dados. Note que um item de dados pode ser "inserido" erroneamente, mas ainda atender as restrições de integridade;

- Fornecer Backup e Restauração: Um SGBD deve fornecer recursos para restauração caso ocorram falhas de hardware ou software. O subsistema de backup e restauração do SGBD é o responsável pela restauração. Por exemplo, se o sistema de computador falhar no meio da execução de um programa que esteja realizando uma alteração complexa na base de dados, o subsistema de restauração é responsável por assegurar que a base de dados seja restaurada ao estado anterior ao início da execução do programa. Alternativamente, o subsistema de restauração poderia assegurar que o programa seja reexecutado a partir do ponto em que havia sido interrompido.

## 2.4 Vantagens Adicionais da Abordagem da Base de Dados

- Potencial para obrigar a Padronização: a abordagem de base de dados permite que o DBA defina e obrigue a padronização entre os usuários da base de dados em grandes organizações. Isso facilita a comunicação e a cooperação entre vários departamentos, projetos e usuários. Padrões podem ser definidos para formatos de nomes, elementos de dados, telas, relatórios, terminologias, etc. O DBA pode obrigar a padronização em um ambiente de base de dados centralizado, muito mais facilmente que em um ambiente onde cada usuário ou grupo tem o controle de seus próprios arquivos e programas de aplicação;
- Flexibilidade: mudanças na estrutura de uma base de dados podem ser necessárias devido a mudanças nos requisitos. Por exemplo, um novo grupo de usuários pode surgir com necessidade de informações adicionais, ainda não disponíveis na base de dados. Alguns SGBD's permitem que tais mudanças na estrutura da base de dados sejam realizadas sem afetar a maioria dos programas de aplicações existentes;
- Redução do Tempo de Desenvolvimento de Aplicações: uma das principais características de venda da abordagem de base de dados é o tempo reduzido para o desenvolvimento de novas aplicações, como a recuperação de certos dados da base de dados para a impressão de novos relatórios. Projetar e implementar uma nova base de dados pode tomar mais tempo do que escrever uma simples aplicação de arquivos especializada. Porém, uma vez que a base de dados esteja em uso, geralmente o tempo para se criar novas aplicações, usando-se os recursos de um SGBD, é bastante reduzido. O tempo para se desenvolver uma nova aplicação em um SGBD é estimado em 1/4 a 1/6 do que o tempo de desenvolvimento, usando-se apenas o sistema de arquivos tradicional, devido às facilidades de interfaces disponíveis em um SGBD;

- Disponibilidade de Informações Atualizadas: tão logo um usuário modifique uma base de dados, todos os outros usuários "sentem" imediatamente esta modificação. Esta disponibilidade de informações atualizadas é essencial para muitas aplicações, tais como sistemas de reservas de passagens aéreas ou bases de dados bancárias. Isso somente é possível devido ao subsistema de controle de concorrência e restauração do SGBD;
- Economia de Escala: a abordagem de SGBD's permite a consolidação de dados e de aplicações reduzindo-se, desse modo, o desperdício em atividades redundantes de processamento em diferentes projetos ou departamentos. Isto possibilita à organização como um todo investir em processadores mais poderosos, e periféricos de armazenamento e de comunicação mais eficientes.

## 2.5 Quando não utilizar um SGBD

- bases de dados e aplicações simples, bem definidas sem expectativa de mudança;
- existem restrições de tempo que não podem ser satisfeitas em SGBD's;
- não há necessidade de acesso multiusuário.

## 2.6 Profissionais e Atividades envolvidas em um SGBD

- Administrador da Base de Dados: em qualquer organização onde muitas pessoas compartilham muitos recursos, existe a necessidade de um administrador chefe para supervisionar e gerenciar estes recursos. Num ambiente de base de dados, o recurso primário é a própria base de dados e os recursos secundários são o próprio SGBD e softwares relacionados. A administração desses recursos é de responsabilidade do DBA ("Database *Administrator*"). O DBA é responsável por autorizar acesso à base de dados e coordenar e monitorar seu uso. O DBA é responsável por problemas, tais como, quebra de segurança ou baixo desempenho. Em grandes organizações, o DBA é auxiliado por técnicos;
- Projetistas da Base de Dados: os projetistas de base de dados têm a responsabilidade de identificar os dados a serem armazenados na Base de Dados e escolher estruturas apropriadas para representar e armazenar tais dados. Estas tarefas são geralmente executadas antes que a base de dados seja utilizada. É responsabilidade destes projetistas obter os requisitos necessários dos futuros usuários da base. Tipicamente, os projetistas interagem com cada grupo de usuários em potencial e definem visões da base de dados para adequar os requisitos e processamentos de cada grupo. Estas visões são então analisadas e, posteriormente, integradas para que, ao final, o projeto da base de dados possa ser capaz de dar subsídio aos requisitos de todos os grupos de usuários;
- Analistas de Sistemas e Programadores de Aplicação:
  - ⇒ analistas de sistemas determinam os requisitos de usuários finais, especialmente dos usuários comuns, e desenvolvem especificações das transações para atender a estes requisitos;
  - ⇒ programadores de aplicações implementam estas especificações produzindo programas e, então, testam, depuram, documentam e mantêm estes programas. Analistas e programadores devem estar familiarizados com todas as capacidades fornecidas pelo SGBD para desempenhar estas tarefas.
- Usuários Finais: existem profissionais que precisam ter acesso à base de dados para consultar, modificar e gerar relatórios. A base de dados existe para estes usuários. Existem algumas categorias de usuários finais:
  - ⇒ usuários ocasionais: ocasionalmente fazem acesso à base de dados, mas eles podem necessitar de diferentes informações a cada vez que fazem acesso. Eles podem usar uma linguagem de consulta sofisticada para especificar suas requisições e são, tipicamente, gerentes de médio ou alto-nível;
  - ⇒ usuários comuns ou paramétricos: estes usuários realizam operações padrões de consultas e atualizações, chamadas TRANSAÇÕES PERMITIDAS, que foram cuidadosamente programadas e testadas. Estes usuários constantemente realizam recuperações e modificações na base de dados;
  - ⇒ usuários sofisticados: incluem engenheiros, analistas de negócios e outros que procuraram familiarizar-se com as facilidades de um SGBD para atender aos seus complexos requisitos;
- Profissionais de Apoio:
  - ⇒ Projetistas e Implementadores de SGBD
  - ⇒ Desenvolvedores de Ferramentas
  - ⇒ Operadores de Manutenção

# 3 Conceitos e Arquiteturas de SGBD's

## 3.1 Modelos de Dados, Esquemas e Instâncias

Uma das características fundamentais da abordagem de base de dados é que ela fornece algum nível de abstração de dados, pela omissão de detalhes de armazenamento de dados que não são necessários para a maioria dos usuários. O modelo de dados é a principal ferramenta que fornece esta abstração. Um Modelo de Dados é um conjunto de conceitos que podem ser usados para descrever a estrutura de uma base de dados. Por estrutura de uma base de dados entende-se os tipos de dados, relacionamentos e restrições pertinentes aos dados. Muitos modelos de dados também definem um conjunto de operações para especificar como recuperar e modificar a base de dados.

### 3.1.1 Categorias de Modelos de Dados

Muitos modelos de dados têm sido propostos. Pode-se classificar os modelos de dados baseando-se nos tipos de conceitos que fornecem para descrever a estrutura da base de dados. Modelos de Dados Conceituais ou de Alto-Nível fornecem conceitos próximos à percepção dos usuários. Já os Modelos de Dados Físicos ou de Baixo-Nível fornecem conceitos que descrevem os detalhes de como os dados são armazenados no computador.

Modelos de alto-nível utilizam conceitos tais como Entidades, Atributos e Relacionamentos. Uma entidade é um objeto que é representado na base de dados. Um atributo é uma propriedade que descreve algum aspecto de um objeto. Relacionamentos entre objetos são facilmente representados em modelos de dados de alto-nível, que são algumas vezes chamados de Modelos Baseados em Objeto devido, principalmente, a sua característica de descreverem objetos e seus relacionamentos.

Modelos de Dados de Baixo-Nível descrevem como os dados são armazenados no computador, representando informações em formato de registros, ordem dos registros e caminho de acesso. Um Caminho de Acesso é uma estrutura de que facilita a busca de um registro particular na base de dados.

### 3.1.2 Esquemas e Instâncias

Em qualquer modelo de dados é importante distinguir entre a descrição da base de dados e a base de dados propriamente dita. A descrição de uma base de dados é chamada Esquema da Base de Dados. Um esquema de base de dados é especificado durante o projeto da base de dados, sendo que a expectativa de mudanças não é grande. A forma de visualização de um esquema é chamada Diagrama do Esquema. Muitos modelos de dados têm certas convenções para, diagramaticamente, mostrar esquemas especificados no modelo.

Os dados atualmente existentes em uma base de dados podem mudar com relativa freqüência. Os dados da base de dados em um particular momento do tempo são chamados Instâncias da Base de Dados (ou Ocorrências ou Estados). A base-esquema é algumas vezes chamada de Base-Intencional e uma instância é chamada de Base-Extensional do esquema.

## 3.2 Arquitetura e Independência de Dados de SGBD's

A arquitetura mais difundida na literatura é a Arquitetura "Three-Schema" (também conhecida como arquitetura ANSI/SPARC), proposta por Tsichritzis & Klug em 1978.

A meta desta arquitetura, exibida na Figura 3.1, é separar as aplicações de usuários da base de dados física. Nesta arquitetura, esquemas podem ser definidos em três níveis:

1. O nível interno tem um esquema interno que descreve a estrutura de armazenamento físico da base de dados. O esquema interno usa um modelo de dados físico e descreve todos os detalhes de armazenamento de dados e caminhos de acesso à base de dados;
2. O nível conceitual tem um esquema conceitual que descreve a estrutura de toda a base de dados. O esquema conceitual é uma descrição global da base de dados, que omite detalhes da estrutura de armazenamento físico e se concentra na descrição de entidades, tipos de dados, relacionamentos e restrições. Um modelo de dados de alto-nível ou um modelo de dados de implementação podem ser utilizados neste nível.
3. O nível externo ou visão possui esquemas externos ou visões de usuários. Cada esquema externo descreve a visão da base de dados de um grupo de usuários da base de dados. Cada visão descreve, tipicamente, a parte da base de dados que um particular grupo de usuários está interessado e esconde deste o restante da base de dados. Um modelo de dados de alto-nível ou um modelo de dados de implementação podem ser usados neste nível.

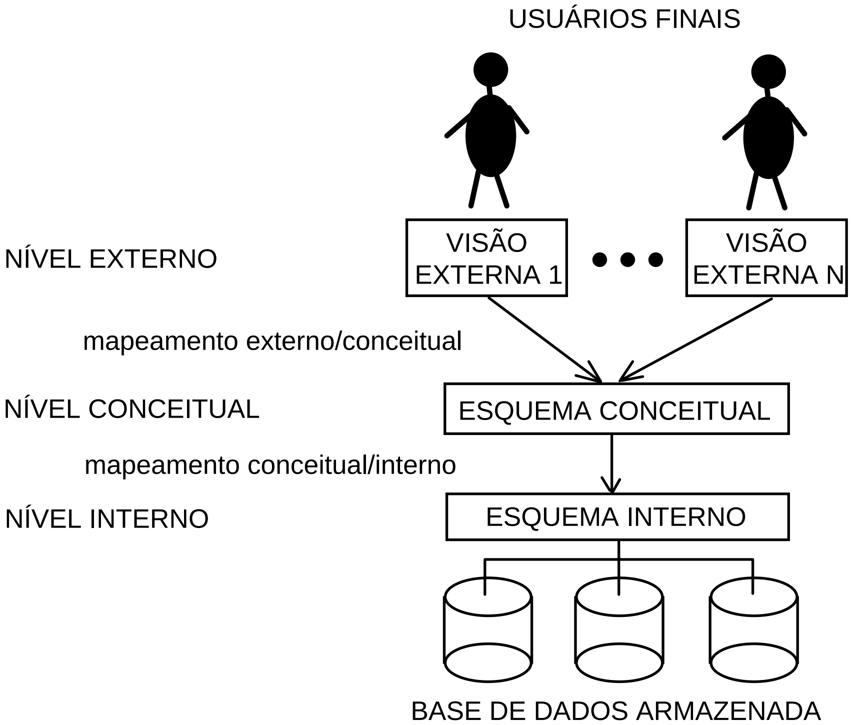



**Figura 3.1 – Arquitetura Three-Schema**

Muitos SGBD's não separam os três níveis completamente. Pode acontecer que alguns SGBD's incluam detalhes do nível interno no esquema conceitual. Em muitos SGBD's que permitem visões, os esquemas externos são especificados com o mesmo modelo de dados usado no nível conceitual. Note que os três esquemas são apenas descrições dos dados.

A arquitetura "three-schema" pode ser utilizada para explicar conceitos de independência de dados, que podem ser definidos como a capacidade de alterar o esquema de um nível sem ter que alterar o esquema no próximo nível superior. Dois tipos de independência de dados podem ser definidos:

- Independência Lógica de Dados: É a capacidade de alterar o esquema conceitual sem ter que mudar os esquemas externos ou programas de aplicação. Pode-se mudar o esquema conceitual para expandir a base de dados, com a adição de novos tipos de registros (ou itens de dados), ou reduzir a base de dados removendo um tipo de registro. Neste último caso, esquemas externos que se referem apenas aos dados remanescentes não devem ser afetados;

- Independência Física de Dados: É a capacidade de alterar o esquema interno sem ter que alterar o esquema conceitual externo. Mudanças no esquema interno podem ser necessárias devido a alguma reorganização de arquivos físicos para melhorar o desempenho nas recuperações e/ou modificações. Após a reorganização, se nenhum dado foi adicionado ou perdido, não haverá necessidade de modificar o esquema conceitual.

## 3.3 Linguagens de Base de Dados

- Linguagem de Definição de Dados ("Data Definition Language" - DDL): é utilizada pelo DBA e projetistas de base de dados para definir seus esquemas. O SGBD tem um compilador para processar descrições em DDL e construir a descrição do esquema armazenado no catálogo;
- Linguagem de Manipulação de Dados ("Data Manipulation Language" - DML): uma vez que o esquema é compilado e a base de dados preenchida com dados, os usuários têm que ter algum modo de manipular os dados. Manipulações comuns como recuperação, inserção, remoção e modificação de dados são realizadas pela DML.

# 4 Modelagem de Dados Usando o Modelo Entidade-Relacionamento (MER)

O MER é um modelo de dados conceitual de alto-nível, ou seja, seus conceitos foram projetados para serem compreensíveis a usuários, descartando detalhes de como os dados são armazenados.

Atualmente, o MER é usado principalmente durante o processo de projeto da base de dados. Existem expectativas para que uma classe de SGBD's baseados diretamente no MER esteja disponível no futuro.

## 4.1 Modelo de Dados Conceitual de Alto-Nível e Projeto de Base Dados

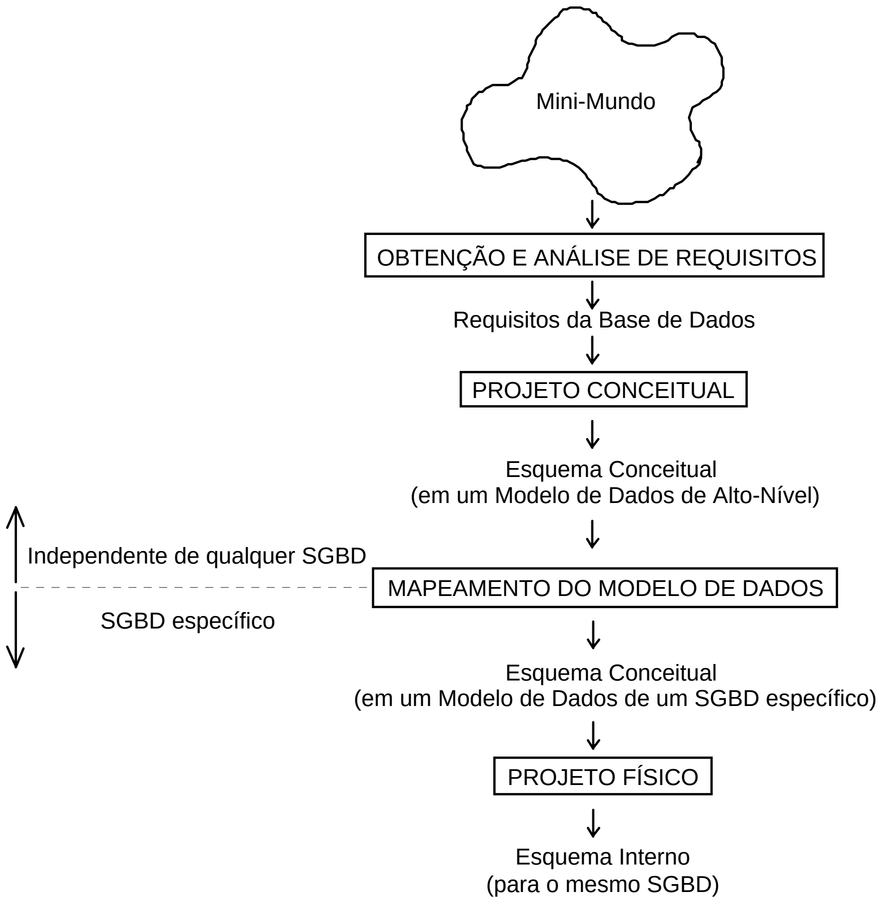



**Figura 4.1 – Esquema geral de modelagem de dados usando MER**

## 4.2 Um Exemplo

Neste exemplo, é descrita uma base de dados COMPANHIA que será utilizada para ilustrar o processo de projeto de base de dados. São listados os requisitos da base de dados e criado o seu esquema conceitual passo-a-passo ao mesmo tempo em que são introduzidos os conceitos de modelagem usando o MER. A base de dados COMPANHIA armazena os dados dos empregados, departamentos e projetos. Supõe-se que após a Obtenção e Análise dos Requisitos, os projetistas da base de dados produziram a seguinte descrição do mini-mundo - parte da companhia a ser representada na base de dados:

- A companhia é organizada em departamentos. Cada departamento tem um nome, um número e um empregado que gerencia o departamento. Armazena-se a data de início que o empregado começou a gerenciar o departamento. Um departamento pode ter diversas localizações;
- Um departamento controla inúmeros projetos, sendo que cada um tem um nome, um número e uma localização;
- Do empregado armazena-se o nome, o número do seguro social, endereço, salário, sexo e data de nascimento. Todo empregado é associado a um departamento, mas pode trabalhar em diversos projetos, que não são necessariamente controlados pelo mesmo departamento. Armazena-se, também, o número de horas que o empregado trabalha em cada projeto. Mantém-se, ainda, a indicação do supervisor direto de cada projeto;
- Os dependentes de cada empregado são armazenados para propósito de garantir os benefícios do seguro. Para cada dependente será armazenado o nome, sexo, data de nascimento e o relacionamento com o empregado.

## 4.3 Conceitos do Modelo Entidade-Relacionamento

### 4.3.1 Entidades e Atributos

O objeto básico que o MER representa é a entidade. Uma entidade é algo do mundo real que possui uma existência independente. Uma entidade pode ser um objeto com uma existência física - uma pessoa, carro ou empregado - ou pode ser um objeto com existência conceitual - uma companhia, um trabalho ou um curso universitário. Cada entidade tem propriedades particulares, chamadas atributos, que a descrevem. Por exemplo, uma entidade EMPREGADO pode ser descrita pelo seu nome, o trabalho que realiza, idade, endereço e salário. Uma entidade em particular terá um valor para cada um de seus atributos. Os valores de atributos que descrevem cada entidade ocupam a maior parte dos dados armazenados na base de dados.

A Figura 4.2 ilustra duas entidades. A entidade $e_1$, EMPREGADO, tem quatro atributos: **Nome**, **Endereço**, **Data de nascimento** e **Telefone residencial**. Os seus valores são: “João da Silva”, “Rua Goiás 711, São Paulo, SP, 1301100”, “31/07/1973” e “713-749”, respectivamente. A entidade $c_1$, COMPANHIA, tem três atributos: **Nome**, **Sede** e **Presidente**. Seus valores são: “Cooper Sugar”, “Ribeirão Preto” e “João da Silva”.

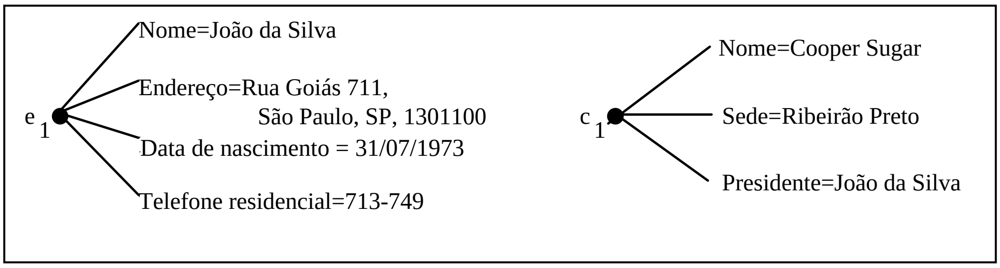



**Figura 4.2 – Exemplo de entidades e seus respectivos atributos**

Alguns atributos podem ser divididos em subpartes com significados independentes. Por exemplo, **Endereço** da entidade $e_1$ pode ser dividido em **Endereço da Rua**, **Cidade**, **Estado** e **CEP**. Um atributo que é composto de outros atributos mais básicos é chamado composto. Já, atributos que não são divisíveis são chamados simples ou atômicos. Atributos compostos podem formar uma hierarquia, conforme pode ser observado no exemplo da Figura 4.3.

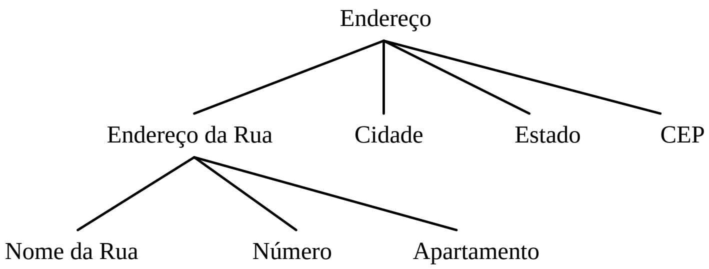



**Figura 4.3 – Um exemplo de atributo composto**

Atributos compostos são úteis quando os usuários referenciam o atributo composto como uma unidade e, em outros momentos, referenciam especificamente a seus componentes. Se o atributo composto for sempre referenciado como um todo, não existe razão para subdividi-lo em componentes elementares.

Muitos atributos têm apenas um único valor. Tais atributos são chamados atributos uni-valorados (exemplo, **Data de nascimento** da entidade $\mathbf{e_1}$). Em outros casos, um atributo pode ter um conjunto de valores. Tais atributos são chamados de atributos multivalorados (exemplo, **Telefone residencial** da entidade $\mathbf{e_1}$). Atributos multivalorados podem possuir uma multiplicidade, indicando as quantidades mínima e máxima de valores.

Em alguns casos, dois ou mais atributos são relacionados. Por exemplo, **Idade** e **Data de Nascimento** de uma pessoa. Para uma entidade pessoa em particular, a **Idade** pode ser determinada a partir da data atual e da **Data de Nascimento**. Atributos como **Idade** são chamados atributos derivados[3]. Alguns valores de atributos podem ser derivados de entidades relacionadas. Por exemplo, um atributo **Número de Empregados** de uma entidade departamento que pode ser calculado contando-se o número de empregados relacionados com o departamento.

Outras situações: uma entidade pode não ter quaisquer valores para um atributo. Por exemplo, o atributo **Apartamento** aplica-se somente àqueles empregados que residam em algum prédio. Para tais situações, um valor especial chamado *null* é criado. O valor *null* pode também ser utilizado para denotar que o valor é desconhecido, como por exemplo, quando o cliente em um cadastro não responde o número do **CEP** da rua onde reside. O significado para o primeiro uso do *null* é "não aplicável" e, para o segundo, "desconhecido".

### 4.3.2 Tipos de Entidades, Conjunto de Valores e Atributos-Chaves

Uma base de dados irá conter normalmente grupos de entidades que são similares. Uma companhia com centenas de empregados pode querer agrupar as informações similares com respeito a empregados. Estas entidades, empregados, compartilham os mesmos atributos, mas cada entidade terá seus próprios valores para cada atributo. Tais entidades similares definem o tipo de entidade, que é um conjunto de entidades que têm os mesmos atributos. Na maioria das bases de dados podem-se identificar muitos tipos de entidades. A Figura 4.4 mostra dois tipos de entidades denominadas EMPREGADO e COMPANHIA, e uma lista de atributos. Algumas instâncias são também ilustradas, juntamente com os seus valores de atributos:

[3] Atributos derivados não necessitam ser armazenados na base de dados, podendo ser calculados por meio de uma consulta.

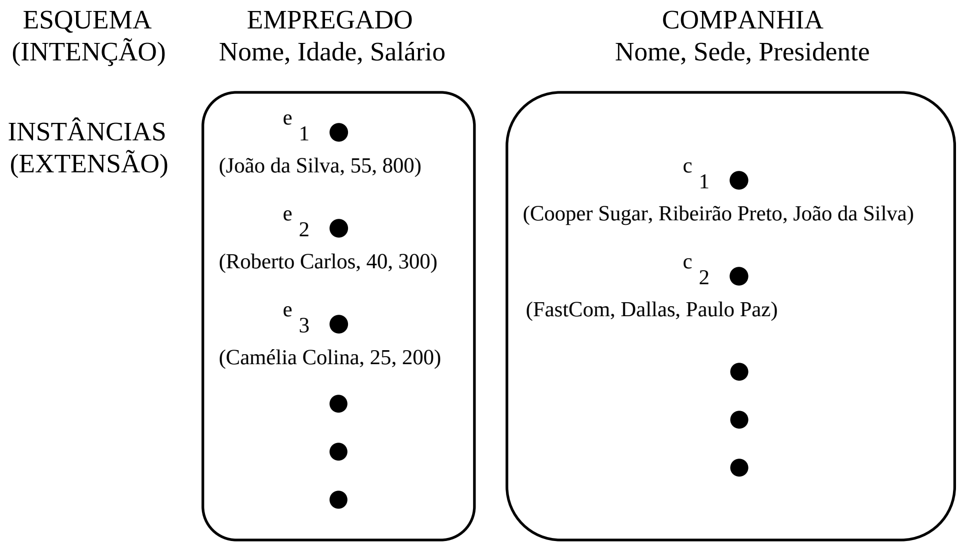



**Figura 4.4 – Exemplo de entidades e suas instâncias**

A descrição do tipo de entidade é chamada **esquema do tipo de entidade** e especifica uma estrutura comum compartilhada por todas as entidades individuais. O esquema especifica o nome do tipo de entidade, o significado de cada um de seus atributos e qualquer restrição que exista sobre entidades individuais. O conjunto de instâncias de entidades em um particular momento do tempo é chamado **extensão** do tipo de entidade. O esquema não é alterado com freqüência porque descreve a estrutura das entidades individuais. A extensão pode mudar com freqüência, por exemplo, quando se adiciona ou remove-se uma entidade de um tipo de entidade.

- Atributo-Chave de um Tipo de Entidade: Uma restrição importante sobre entidades de um tipo de entidade é a restrição de **atributo-chave** ou **unicidade**. Um tipo de entidade tem, normalmente, atributos cujos valores são distintos para cada entidade. Tal atributo é chamado atributo-chave, e o seu valor pode ser usado para identificar cada entidade unicamente. Algumas vezes, um conjunto de atributos pode formar uma chave. Nestes casos, os atributos podem ser agrupados em um atributo composto, que virá a ser um atributo-chave do tipo de entidade.

  A especificação de um atributo-chave para um tipo de entidade significa que a propriedade de unicidade deve valer para quaisquer extensões deste tipo de entidade. Assim, esta restrição proíbe que duas entidades tenham, simultaneamente, o mesmo valor para o atributo-chave.

  Alguns tipos de entidades podem ter mais que um atributo-chave.

## 4.3.3 Relacionamentos, Papéis e Restrições Estruturais

- Tipo e Instância de Relacionamento: Um **tipo de relacionamento** R entre n tipos de entidades $E_1$, $E_2$, ..., $E_n$ é um conjunto de associações entre entidades desses tipos. Diz-se que cada entidade $E_1$, $E_2$, ..., $E_n$ **participa** no tipo de relacionamento R e que as entidades individuais $e_1$, $e_2$, ..., $e_n$ participam na instância do relacionamento $r_i$=($e_1$, $e_2$, ..., $e_n$). O índice i indica que podem existir várias instâncias de relacionamento.

  Por exemplo, considere-se que um tipo de relacionamento TRABALHA-PARA exista entre tipos de entidades EMPREGADO e DEPARTAMENTO. Este relacionamento associa cada

empregado com o departamento em que este trabalha. Cada instância de relacionamento em TRABALHA-PARA associa uma entidade empregado e uma entidade departamento. A Figura 4.5 ilustra este exemplo, onde cada instância de relacionamento $r_i$ é conectada a uma entidade empregado e a uma entidade departamento. No mini-mundo representado nesta Figura, os empregados $e_1$, $e_3$ e $e_6$ trabalham para o departamento $d_1$; $e_2$ e $e_4$ trabalham para $d_2$; e $e_5$ e $e_7$ trabalham para $d_3$.

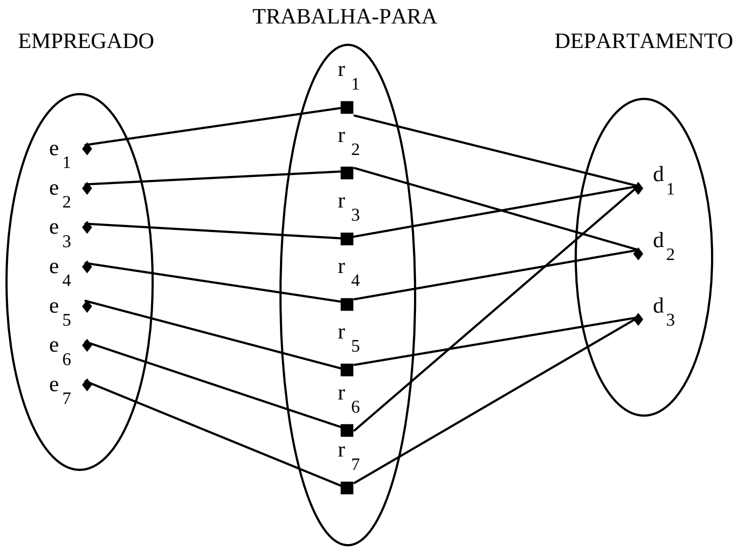



**Figura 4.5 – Exemplo de um Relacionamento Binário**

- O Grau de um Tipo de Relacionamento: O grau de um tipo de relacionamento indica o número de tipos de entidades participantes. Assim, o tipo de relacionamento TRABALHA-PARA é de grau dois. Um tipo de relacionamento de grau dois é chamado **binário** e de grau três de **ternário**. Um exemplo de um tipo de relacionamento ternário é FORNECE, ilustrado na Figura 4.6. Cada instância de relacionamento $r_i$ associa três entidades - um fornecedor s, uma peça p e um projeto j - onde o fornecedor s fornece a peça p para o projeto j. Podem existir tipos de relacionamento de qualquer grau, porém é mais freqüente encontrar o tipo de relacionamento de grau dois.

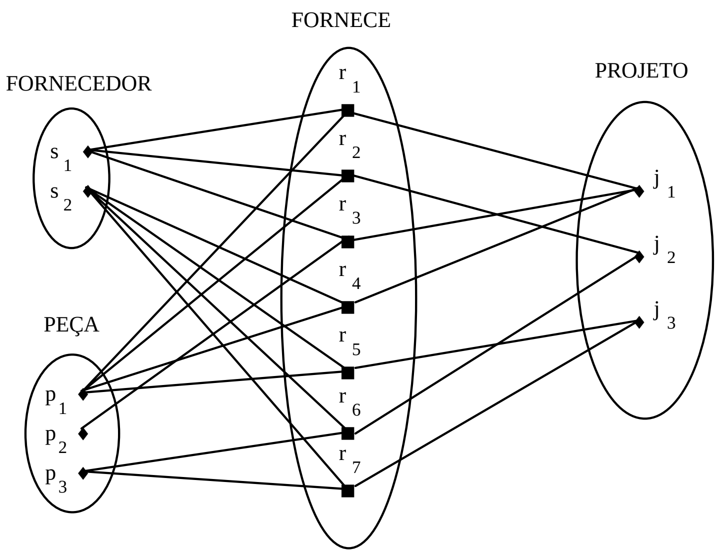



**Figura 4.6 – Exemplo de um Relacionamento Ternário**

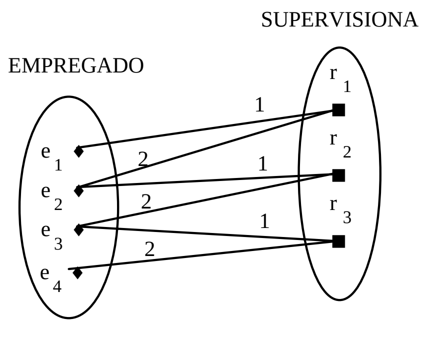
Em geral, um tipo de relacionamento ternário representa mais informação do que três tipos de relacionamentos binários. Por exemplo, considere os três tipos de relacionamentos binários: PODE-FORNECER, USA e FORNECE-ALGO. Supõe-se que:

a. PODE-FORNECER, entre os tipos de entidades FORNECEDOR e PEÇA, possui uma instância (s, p) com o significado: "o fornecedor s pode fornecer a peça p" (para qualquer projeto);
b. USA, entre os tipos de entidades PROJETO e PEÇA, possui uma instância (j, p) com o significado: "o projeto j usa a peça p"; e
c. FORNECE-ALGO, entre os tipos de entidades FORNECEDOR e PROJETO, possui uma instância (s, j) com o significado: "o fornecedor s fornece alguma peça para o projeto j".

A existência dessas três instâncias de relacionamentos (s, p), (j, p) e (s, j) em PODE-FORNECER, USA e FORNECE-ALGO, respectivamente, não necessariamente implica que uma instância (s, j, p) exista no tipo de relacionamento ternário FORNECE. Isto tem sido chamado **armadilha de conexão**.

· Relacionamento como Atributo: Convém, algumas vezes, pensar em um tipo de relacionamento em termos de atributos. Considere-se o tipo de relacionamento TRABALHA-PARA discutido anteriormente. Pode-se pensar em colocar um atributo chamado Departamento no tipo de entidade EMPREGADO onde o valor deste atributo em cada entidade empregado é a entidade departamento em que ele trabalha. Quando se pensa em um tipo de relacionamento binário como atributo, existem duas alternativas: Departamento como atributo do tipo de entidade EMPREGADO ou Empregado como atributo do tipo de entidade DEPARTAMENTO. Neste último caso, o atributo Empregado é um atributo multivalorado, onde os valores pertencem ao tipo de entidade EMPREGADO. Qualquer uma dessas alternativas é representada pelo tipo de relacionamento TRABALHA-PARA.

· Nomes de Papéis e Relacionamentos Recursivos: Cada tipo de entidade que participa de um tipo de relacionamento possui um **papel** específico no relacionamento. O **nome do papel** indica o papel que uma entidade de um tipo de entidade tem para cada instância de relacionamento. Por exemplo, no tipo de relacionamento TRABALHA-PARA, EMPREGADO tem o papel empregado ou trabalhador e DEPARTAMENTO tem o papel de departamento ou empregador. A escolha do nome nem sempre é simples. Para o tipo de relacionamento ternário FORNECE, é difícil encontrar-se um nome. O nome de papel não é exclusividade do tipo de relacionamento onde os tipos de entidades participantes são distintos. Em alguns casos, um mesmo tipo de entidade participa mais que uma vez em um tipo de relacionamento com diferentes papéis. Nesses casos, é essencial identificar os nomes dos papéis a fim de distinguir o significado de cada participação. Tais tipos de relacionamentos são chamados **recursivos**. Para ilustrar, considere a Figura 4.7:




## Figura 4.7 – Exemplo de auto-relacionamento (Papel do relacionamento)

O tipo de relacionamento SUPERVISIONA relaciona um empregado com o seu supervisor, onde ambas entidades são membros do mesmo tipo de entidade EMPREGADO. Assim, o tipo de entidade EMPREGADO participa duas vezes: uma vez no papel de supervisor e outra no papel de supervisionado. Na 4.7 acima, as linhas marcadas com "1" representam

o papel de supervisor e os marcados com "2" representam o papel de supervisionado. Assim, $e_1$ supervisiona $e_2$, $e_2$ supervisiona $e_3$ e $e_3$ supervisiona $e_4$.

- Restrições sobre Tipos de Relacionamentos: Os tipos de relacionamento possuem certas restrições que limitam as combinações possíveis de entidades participando nas instâncias de relacionamento. Estas restrições são determinadas pelas situações do mini-mundo que os relacionamentos representam. Por exemplo, na Figura 4.5, se existir uma regra que um empregado trabalha para apenas um departamento, então esta restrição deve ser descrita no esquema. Pode-se distinguir dois principais tipos de restrições de relacionamento que ocorrem com relativa freqüência: razão de cardinalidade e participação.

A restrição **razão de cardinalidade** especifica a quantidade de instâncias de relacionamento que uma entidade pode participar. No tipo de relacionamento binário TRABALHA-PARA, DEPARTAMENTO:EMPREGADO tem razão de cardinalidade 1:N. Isto significa que cada entidade departamento pode estar relacionada a inúmeras entidades empregado (muitos empregados podem trabalhar para um departamento) mas uma entidade empregado pode estar relacionada a apenas um departamento (um empregado pode trabalhar apenas para um departamento). As razões de cardinalidade mais comuns para tipos de relacionamento binário são 1:1, 1:N e M:N.

Um exemplo de um tipo de relacionamento binário 1:1 pode ser observado na Figura 4.8, entre DEPARTAMENTO e EMPREGADO é GERENCIA, que relaciona uma entidade departamento com o empregado que gerencia esse departamento. Este relacionamento é 1:1 pois sabe-se que um empregado pode gerenciar apenas um departamento e que um departamento pode ter apenas um gerente.

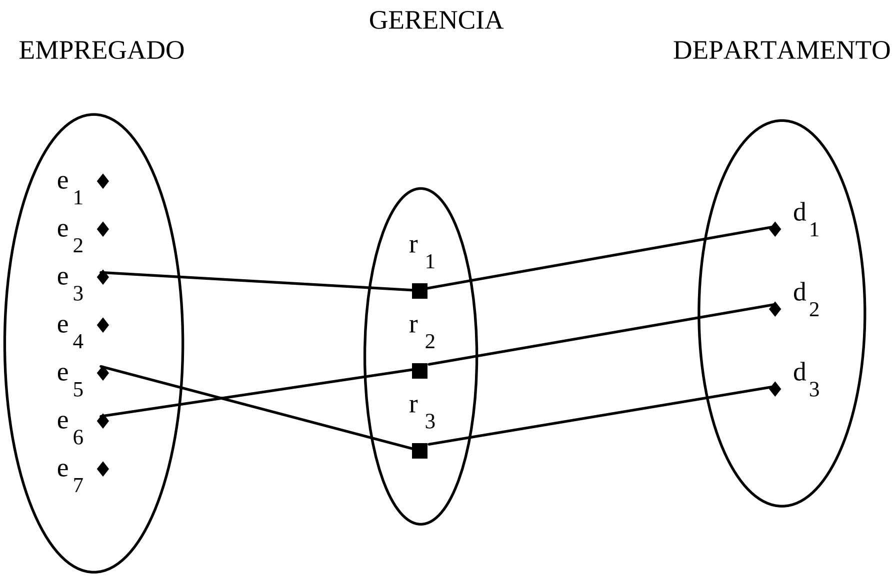



**Figura 4.8 – Exemplo de um relacionamento binário 1:1**

O tipo de relacionamento TRABALHA-EM entre EMPREGADO e PROJETO tem a razão de cardinalidade M:N (Figura 4.9) considerando que um empregado pode trabalhar em diversos projetos e que diversos empregados podem trabalhar em um projeto.

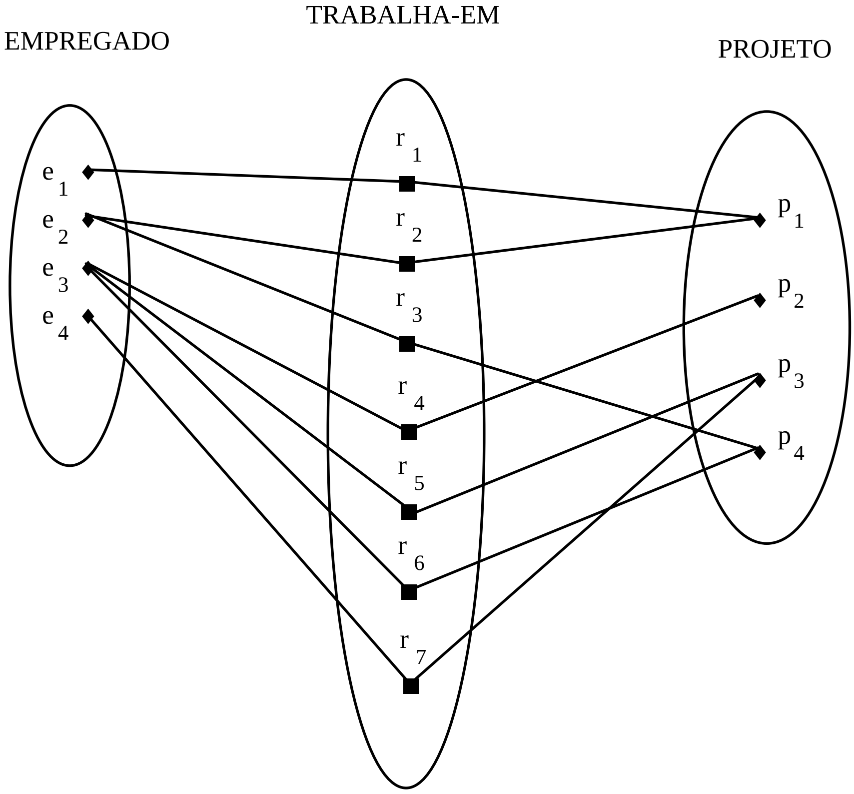



**Figura 4.9 – Exemplo de um relacionamento binário M:N**

A restrição de participação especifica se a existência de uma entidade depende dela estar relacionada com outra entidade através de um relacionamento. Existem dois tipos de restrições de participação: total e parcial. Se uma companhia estabelece a regra de que todo empregado deve trabalhar para um departamento, então uma entidade empregado somente pode existir se ele participar em uma instância de relacionamento TRABALHA-PARA. A participação de EMPREGADO em TRABALHA-PARA é chamada **total**, o que significa que toda entidade empregado deve estar relacionada a uma entidade departamento via TRABALHA-PARA. A restrição de participação total é algumas vezes chamada **dependência existencial**. Não é esperado, na Figura 4.8, que todo empregado gerencie um departamento, assim a participação de EMPREGADO no tipo de relacionamento GERENCIA é **parcial**. Isso significa que algumas entidades, do conjunto de entidades EMPREGADO, poderão estar relacionadas a uma entidade departamento via GERENCIA, mas não necessariamente todas.

As restrições de participação e razão de cardinalidade podem ser derivadas da **restrição estrutural** de um tipo de relacionamento. É simples especificar as restrições estruturais, embora não seja tão intuitiva quanto às restrições de participação e razão de cardinalidade. Pode-se associar um par de números inteiros (min, max) para cada participação de um tipo de entidade E em um tipo de relacionamento R, onde $0 \leq min \leq max$ e $max \geq 1$. Os números indicam que para cada entidade *e* em E, *e* deve participar ao menos min e no máximo max vezes em instâncias de relacionamento de R. Note-se, que se min=0 então a restrição de participação é parcial e se min>0 então a participação é total. A vantagem de usar este método é que ele é mais preciso e pode ser usado facilmente para especificar restrições estruturais para tipos de relacionamentos de qualquer grau.

- Atributos em Tipos de Relacionamento: Os tipos de relacionamento também podem ter atributos da mesma maneira que os tipos de entidades. Por exemplo, pode haver a necessidade de se representar a quantidade de horas semanais trabalhadas por um empregado em um dado projeto. Isto pode ser representado em cada instância de relacionamento TRABALHA-EM na forma de atributo denominado Horas. Um outro

exemplo é o caso de representar a data em que um gerente começou a gerenciar um departamento através de um atributo DataInício para o tipo de relacionamento GERENCIA (Figura 4.8).

Note-se que atributos de tipos de relacionamento 1:1 ou 1:N podem ser incluídos como atributos de um dos tipos de entidades participantes. Por exemplo, o atributo DataInício para o tipo de relacionamento GERENCIA pode ser um atributo tanto de EMPREGADO quanto de DEPARTAMENTO; embora, conceitualmente, ele pertença ao relacionamento GERENCIA. Isso ocorre porque GERENCIA é um relacionamento 1:1. Assim, toda entidade departamento ou empregado participam em apenas uma instância de relacionamento e, dessa forma, o valor do atributo DataInício pode ser representado em uma das entidades participantes.

Para um tipo de relacionamento 1:N, um atributo de relacionamento pode somente ser colocado no tipo de entidade que está do lado N do relacionamento. Por exemplo, na Figura 4.5, se o relacionamento TRABALHA-PARA tiver um atributo DataInício indicando quando um empregado começou a trabalhar para um departamento, este atributo pode ser colocado como atributo de EMPREGADO. Isto acontece porque o relacionamento é 1:N, tal que cada entidade empregado participa apenas uma única vez em uma instância de TRABALHA-PARA. Em ambos os tipos de relacionamento 1:1 e 1:N, a decisão de onde colocar um atributo de relacionamento é determinada subjetivamente pelo projetista de esquemas.

Se o valor de um atributo é determinado pela combinação das entidades participantes em uma instância de relacionamento, e não apenas por uma das entidades, então o atributo deve ser especificado como um atributo de relacionamento. Esta condição aplica-se a atributos de tipos de relacionamentos M:N, porque as entidades dos tipos de entidades participantes podem participar em inúmeras instâncias de relacionamento. Um exemplo disso é o atributo Horas do relacionamento M:N TRABALHA-EM (Figura 4.9). O número de horas que um empregado trabalha em um projeto é determinado pela combinação empregado-projeto e não separadamente.

### 4.3.4 Tipo de Entidade-Fraca

Alguns tipos de entidades podem não ter quaisquer atributos-chaves. Isto implica que não se pode distinguir as entidades porque a combinação dos valores de atributos podem ser idênticas. Tais tipos de entidades são chamadas **tipos de entidades-fracas**. Entidades que pertencem a um tipo de entidade-fraca são identificadas por estarem associadas a entidades específicas de um outro tipo de entidade em combinação com alguns de seus valores de atributos. Este outro tipo de entidade é denominado **proprietário da identificação**, e o tipo de relacionamento que relaciona um tipo de entidade-fraca com o proprietário da identificação é chamado **relacionamento de identificação** do tipo de entidade-fraca. Um tipo de entidade-fraca sempre tem uma restrição de participação total (dependência existencial) com respeito ao seu relacionamento de identificação, porque não é possível identificar uma entidade-fraca sem a correspondente entidade proprietária.

Por exemplo, considere o tipo de entidade DEPENDENTE, relacionado a EMPREGADO, que é usado para representar os dependentes de cada empregado através do relacionamento 1:N. Os atributos de DEPENDENTE são Nome (apenas o primeiro nome do dependente), DataNasc, Sexo e Relação com o empregado (esposa, marido, filho, sogra, etc.). Dois dependentes de empregados distintos podem ter os mesmos valores para os atributos, mesmo assim eles ainda serão entidades distintas. Os dependentes serão identificados como entidades distintas após a determinação da entidade empregado com a qual cada um está relacionado.

Um tipo de entidade-fraca tem uma **chave-parcial**, que é um conjunto de atributos que pode univocamente identificar entidades-fracas relacionadas à mesma entidade proprietária. No exemplo, assume-se que nenhum dependente de um mesmo empregado terá o mesmo nome, então o atributo Nome de DEPENDENTE será a chave-parcial.

Um tipo de entidade-fraca pode, algumas vezes, ser representado como atributo composto e multivalorado. No exemplo, pode-se especificar um atributo composto e multivalorado denominado Dependente para EMPREGADO, onde os atributos componentes são Nome, DataNasc, Sexo e Relação, substituindo-se assim, o tipo de entidade-fraca DEPENDENTE. A escolha de qual representação usar é determinada pelo projetista da base de dados. Um critério usado para se adotar a representação de tipo de entidade-fraca é quando o tipo de entidade-fraca tem muitos atributos ou participa, independentemente, em outros tipos de relacionamentos além do tipo de relacionamento que o identifica.

### 4.3.5 Projeto da Base de Dados COMPANHIA utilizando o MER

Tendo visto os conceitos pertinentes ao MER, pode-se agora especificar os seguintes tipos de relacionamentos extraídos do exemplo apresentado na seção 4.2.

a. GERENCIA (1:1) entre EMPREGADO e DEPARTAMENTO. A participação de EMPREGADO é parcial. A participação de DEPARTAMENTO é total. O atributo DataInício é associado a este tipo de relacionamento.
b. TRABALHA-PARA (1:N) entre DEPARTAMENTO e EMPREGADO. Ambos têm participação total.
c. CONTROLA (1:N) entre DEPARTAMENTO e PROJETO. A participação de PROJETO é total e de DEPARTAMENTO é parcial.
d. SUPERVISIONA (1:N) entre EMPREGADO (no papel de supervisor) e EMPREGADO (no papel de supervisionado). A participação de ambos é parcial, pois nem todo empregado é supervisor e nem todo empregado tem um supervisor.
e. TRABALHA-EM (M:N) entre EMPREGADO e PROJETO com o atributo Horas. Ambos têm participação total.
f. DEPENDENTE-DE (1:N) entre EMPREGADO e DEPENDENTE. É um tipo de relacionamento de identificação para o tipo de entidade-fraca DEPENDENTE. A participação de EMPREGADO é parcial e de DEPENDENTE é total.

Nas Figuras de 4.5 a 4.9 foram representados os tipos de entidades e relacionamentos mostrando suas extensões (as entidades e instâncias de relacionamentos). Em diagramas do MER, ou simplesmente DER, a ênfase está na representação de esquemas ao invés de instâncias. Isso porque o esquema de uma base de dados raramente sofre mudanças, já instâncias podem mudar freqüentemente. Também, o esquema é de fácil visualização por conter menor quantidade de elementos visuais.

## 4.4 Diagrama Entidade-Relacionamento (DER)

A Figura 4.10 ilustra um DER para o esquema da base de dados COMPANHIA. Os tipos de entidades tais como EMPREGADO, DEPARTAMENTO e PROJETO são mostrados em retângulos. Tipos de relacionamentos tais como TRABALHA-PARA, GERENCIA, CONTROLA e TRABALHA-EM são mostrados em losângulos interligados a tipos de entidades participantes. Atributos são mostrados em elipses conectadas a tipos de entidades ou relacionamentos. Os componentes de um atributo composto são também representados em elipses, porém conectadas ao atributo ao qual eles pertencem (atributo Nome de EMPREGADO). Atributos multivalorados são denotados em elipses com linhas duplas (atributo Localização de DEPARTAMENTO). Os atributos-chaves são sublinhados. Atributos derivados em elipses com linhas tracejadas (atributo NumeroDeEmpregados de DEPARTAMENTO).

Os tipos de entidades-fracas são distinguidos por retângulos com linhas duplas e os relacionamentos de identificação por losângulos com linhas duplas (tipo de entidade-fraca DEPENDENTE e tipo de relacionamento de identificação DEPENDENTE-DE). A chave-parcial de um tipo de entidade-fraca é sublinhada com linha tracejada.

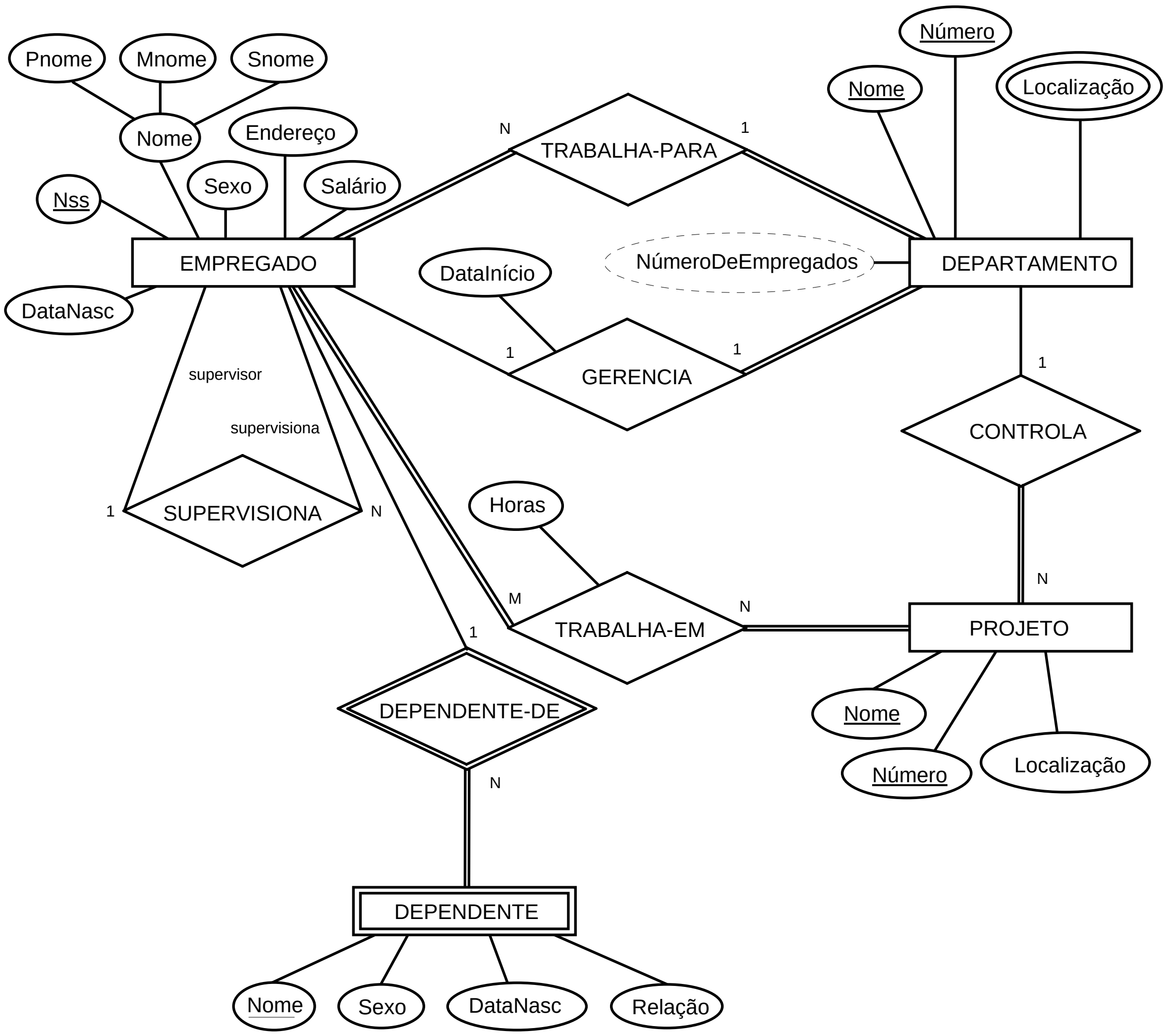



**Figura 4.10 – Diagrama Entidade Relacionamento para o Esquema Companhia**

Na Figura 4.10 são mostradas as razões de cardinalidade para cada tipo de relacionamento binário. A razão de cardinalidade de DEPARTAMENTO:EMPREGADO em GERENCIA é 1:1, para DEPARTAMENTO:EMPREGADO em TRABALHA-PARA é 1:N e M:N para TRABALHA-EM. As restrições de participação parcial são especificadas por linhas simples. As linhas paralelas denotam participação total (dependência existencial).

Na Figura 4.10 foram mostrados os nomes de papéis para o tipo de relacionamento SUPERVISIONA porque o tipo de entidade EMPREGADO ocupa dois papéis neste relacionamento.

Na Figura 4.11 é mostrado o mesmo esquema da Figura 4.10, porém com a utilização da notação alternativa para ilustrar as restrições estruturais de tipos de relacionamentos.

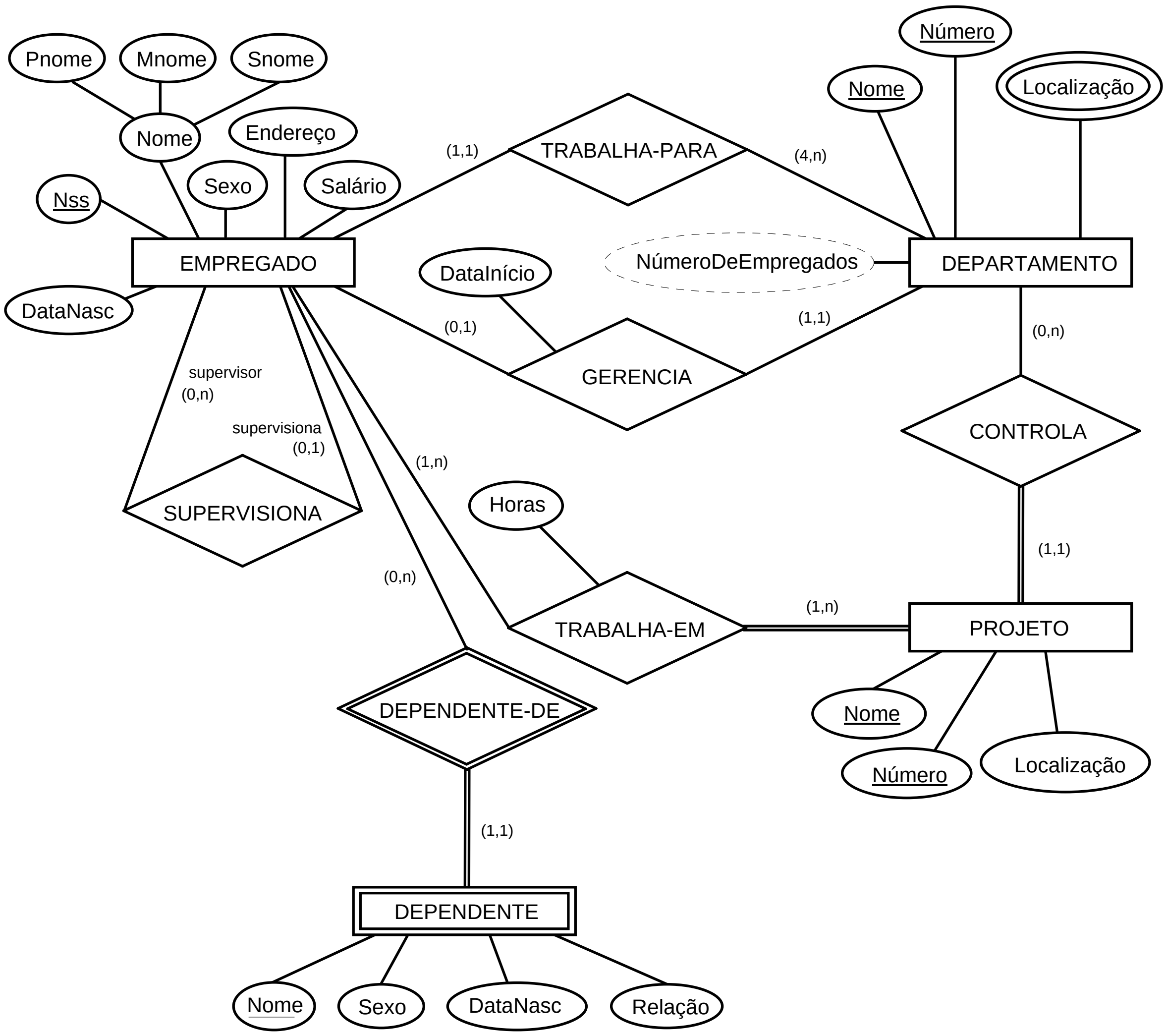



**Figura 4.11 – DER para o Esquema Companhia com notação alternativa**

## 4.5 Tipos de Relacionamentos de Grau maior que Dois

Na Seção 4.3.3 definiu-se **grau** de um tipo de relacionamento como o número de tipos de entidades participantes e chamou-se o tipo de relacionamento de grau dois de **binário** e de grau três de **ternário**. A notação do DER para um tipo de relacionamento ternário é mostrada na Figura 4.13(a), que mostra o esquema para o tipo de relacionamento FORNECE que foi mostrado no nível de extensão na Figura 4.6. Em geral, um tipo de relacionamento R de grau n irá ter n arestas no DER, cada um conectando R para cada tipo de entidade participante.

A Figura 4.13(b) mostra um DER para os tipos de relacionamento binário PODE-FORNECER, USA E FORNECE-ALGO. Como visto na Seção 4.3.3, estes três tipos de relacionamento binário não são equivalentes ao tipo de relacionamento ternário FORNECE. É freqüentemente difícil decidir se um determinado relacionamento deve ser representado como um tipo de relacionamento de grau n ou dividido em muitos tipos de relacionamentos de menor grau. O projetista da base de dados deve guiar-se pela semântica, ou pelo significado da situação particular que estiver representando, para decidir qual alternativa adotar.

Um outro exemplo é mostrado na Figura 4.13(c). O tipo de relacionamento ternário FORNECE representa a informação sobre os instrutores que oferecem cursos em um dado semestre. Assim, ele possui uma instância de relacionamento (i, s, c) onde o instrutor i oferece o curso c

durante o semestre s. Os três tipos de relacionamento binários mostrados na Figura 4.13(c) têm os seguintes significados:

a. PODE-DAR: relaciona os instrutores que *podem dar* um dado curso.
b. DÁ: relaciona os instrutores que dão *algum curso* num dado semestre.
c. OFERECIDO: relaciona os cursos oferecidos num dado semestre *por algum professor*.

Em geral, os relacionamentos binários e ternários representam informações diferentes, mas certas restrições podem ser feitas entre estes relacionamentos. Por exemplo, uma instância de relacionamento (i, s, c) somente irá existir em OFERECE se a instância (i, s) em DÁ, a instância (s, c) em OFERECIDO e a instância (i, c) em PODE-DAR existirem. No entanto, o inverso não é sempre verdade; pode ser que existam instâncias (i, s), (s, c) e (i, c) nos três tipos de relacionamentos binários e, mesmo assim, não existir nenhuma instância (i, s, c) em OFERECE. Sobre certas restrições adicionais (i, s, c) pode existir, por exemplo, se o tipo de relacionamento PODE-DAR for 1:1, ou seja, se um instrutor somente puder dar aulas em apenas um curso e um curso puder ter apenas um instrutor. O projetista da base de dados deve analisar cada situação específica para decidir qual tipo de relacionamento binário e ternário irá necessitar.

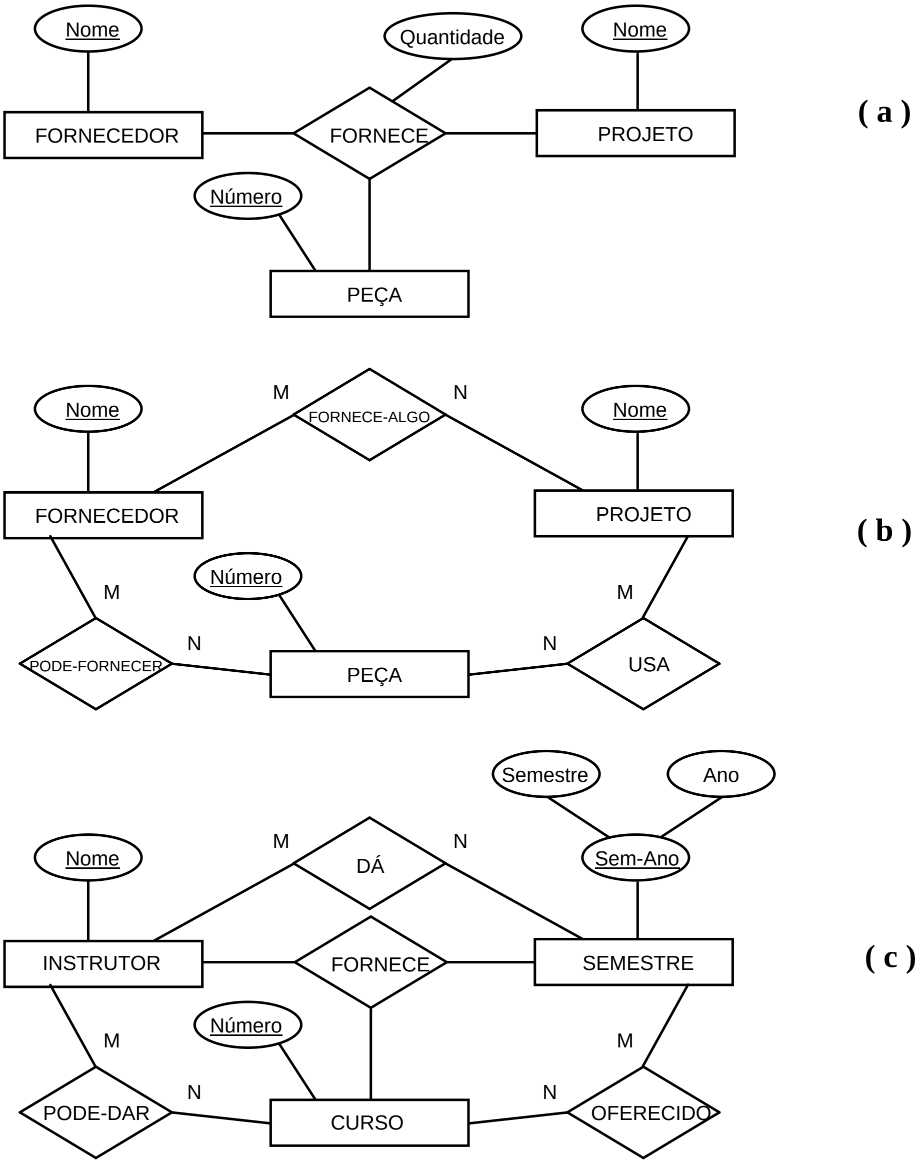



**Figura 4.13 – Comparação entre relacionamentos binários e ternários.**

Note-se que é possível ter um tipo de entidade-fraca com um tipo de relacionamento de identificação ternário (ou n-ário). Neste caso, o tipo de entidade-fraca pode ter muitos tipos de entidades proprietárias da identificação. Um exemplo é mostrado na Figura 4.14. As razões de cardinalidade também são aplicáveis em tipos de relacionamentos n-ários.

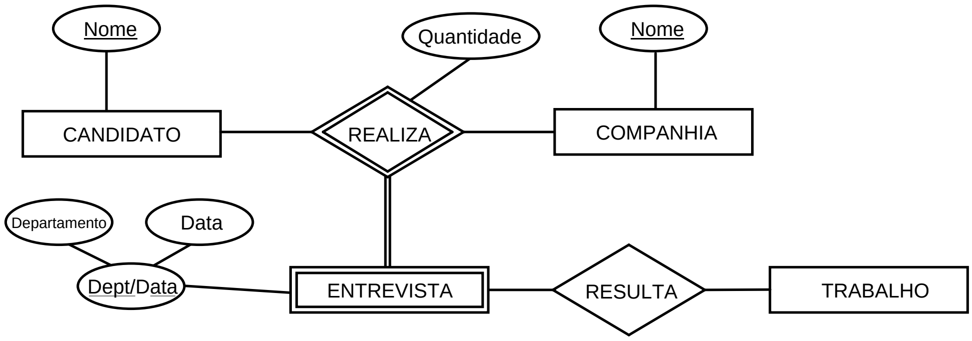
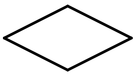
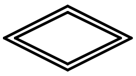
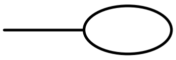
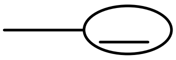
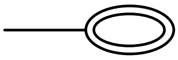
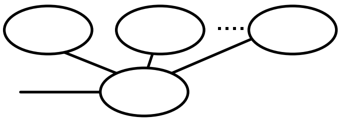
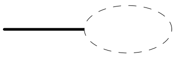
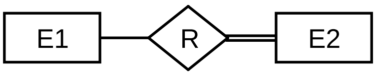
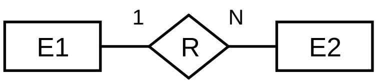
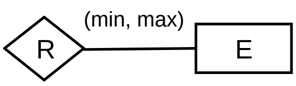



**Figura 4.14 – Exemplo de relacionamento ternário com entidade fraca.**

## Sumário da Notação do Diagrama Entidade-Relacionamento (DER)

| Símbolo | Significado |
| --- | --- |
|  | Tipo de Entidade |
|  | Tipo de Entidade-Fraca |
|  | Tipo de Relacionamento |
|  | Tipo de Relacionamento Identificador |
|  | Atributo |
|  | Atributo-Chave |
|  | Atributo Multivalorado |
|   | Atributo Composto |
|  | Atributo Derivado |
|   | Participação Total de E2 em R |
|   | Razão de Cardinalidade 1:N para E1:E2 em R |
|   | Restrição Estrutural (min, max) na participação de E em R |

## 4.6 Questões para a Síntese

1. Discuta o papel de um modelo de dados de alto-nível no processo de projeto de base de dados.
2. Cite alguns casos onde o valor *null* pode ser aplicado.
3. Defina os seguintes termos: entidade, atributo, valor de atributo, instância de relacionamento, atributo composto, atributo multivalorado, atributo derivado e atributo-chave.
4. O que é um tipo de entidade? Descreva as diferenças entre entidade e tipo de entidade.
5. O que é um tipo de relacionamento? Descreva as diferenças entre instância e tipo de relacionamento.
6. Quando é necessário utilizar nome de papéis na descrição de tipos de relacionamentos?
7. Descreva as duas alternativas para especificar as restrições estruturais sobre tipos de relacionamentos. Quais são as vantagens e desvantagens de cada uma?
8. Sobre quais condições pode um atributo de um tipo de relacionamento binário ser promovido a um atributo de um dos tipos de entidades participantes?
9. Sobre quais condições um tipo de relacionamento pode se tornar um atributo de um tipo de entidade?
10. Qual o significado de um tipo de relacionamento recursivo? Dê alguns exemplos disso.
11. Quando o conceito de entidade-fraca é útil na modelagem de dados? Defina os termos: tipo de entidade proprietário, tipo de relacionamento de identificação e chave-parcial.
12. Um tipo de relacionamento de identificação pode ter grau maior que dois?
13. Discuta as condições em que um tipo de relacionamento ternário pode ser representado por um número de tipos de relacionamentos binários.

# 5 O Modelo de Dados Relacional

O Modelo de Dados Relacional foi introduzido por Codd (1970). Entre os modelos de dados de implementação, o modelo relacional é o mais simples, com estrutura de dados uniforme, e também o mais formal.

## 5.1 Conceitos do Modelo Relacional

O modelo de dados relacional representa os dados da base de dados como uma coleção de relações. Informalmente, cada relação pode ser entendida como uma tabela ou um simples arquivo de registros.

Por exemplo, a base de dados de arquivos representada pela Figura 5.1, é considerada estando no modelo relacional. Porém, existem diferenças importantes entre relações e arquivos.

ESTUDANTE

| Nome | Número | Classe | Departamento |
|---|---|---|---|
| Soares | 17 | 1 | DCC |
| Botelho | 8 | 2 | DCC |

CURSO

| Nome | Número | Créditos | Departamento |
|---|---|---|---|
| Introd. Ciências de Comp. | DCC1310 | 4 | DCC |
| Estrutura de Dados | DCC3320 | 4 | DCC |
| Matemática Discreta | MAT2410 | 4 | MAT |
| Base de Dados | DCC3380 | 4 | DCC |

PRÉ-REQUISITO

| Número | Pré-requisito |
|---|---|
| DCC3380 | DCC3320 |
| DCC3380 | MAT2410 |
| DCC3320 | DCC1310 |

SEÇÃO

| Número | Curso | Semestre | Ano | Professor |
|---|---|---|---|---|
| 85 | MAT2410 | 1 | 86 | Kotaro |
| 92 | DCC1310 | 1 | 86 | Alberto |
| 102 | DCC3320 | 2 | 87 | Kleber |
| 112 | MAT2410 | 1 | 87 | Carlos |
| 119 | DCC1310 | 1 | 87 | Alberto |
| 135 | DCC3380 | 1 | 87 | Souza |

HISTÓRICO

| NúmeroEstudante | NúmeroSeção | Nível |
|---|---|---|
| 17 | 112 | B |
| 17 | 119 | C |
| 8 | 85 | A |
| 8 | 92 | A |
| 8 | 102 | B |
| 8 | 135 | A |

**Figura 5.1 – Exemplo de uma base de dados relacional**

Quando uma relação é vista como uma **tabela** de valores, cada linha representa uma coleção de valores relacionados. Esses valores podem ser interpretados como um fato que descreve uma entidade ou uma instância de relacionamento. O nome da tabela e os nomes das colunas são usados para ajudar a interpretar o significado dos valores em cada linha da tabela. Por

exemplo, na Figura 5.1 anterior, a primeira tabela é chamada ESTUDANTE porque cada linha representa o fato sobre uma particular entidade estudante. Os nomes das colunas - Nome, Número, Classe, Departamento - especificam como interpretar os valores em cada linha, baseando-se nas colunas em que cada um se encontra. Todos os valores de uma mesma coluna são, normalmente, do mesmo tipo.

Na terminologia de base de dados relacional, a linha é chamada de **tupla**, a coluna é chamada de **atributo** e a tabela de **relação**. O tipo de dado que especifica o tipo dos valores que podem aparecer em uma coluna é chamado de **domínio**.

Uma **relação esquema** R, denotada por $R(A_1, A_2, ..., A_n)$, é um conjunto de atributos $R=\{A_1, A_2, ..., A_n\}$. Cada atributo $A_i$ indica o nome do papel de algum domínio D na relação esquema R. D é chamado **domínio** de $A_i$ e denotado por $dom(A_i)$. Uma relação esquema é utilizada para descrever uma relação e R é o nome dessa relação. O **grau de uma relação** é o número de atributos da relação.

Considere o exemplo de uma relação esquema de grau 7, que descreve estudantes universitários:

ESTUDANTE(Nome, NSS, Telefone, Endereço, TelComercial, Anos, MPA)

Nesta relação esquema, ESTUDANTE é o nome da relação esquema, que tem 7 atributos. Pode-se especificar alguns domínios para atributos da relação ESTUDANTE:

dom(Nome)=Nomes
dom(NSS)=Número do seguro social
dom(Telefone)=Número de telefone nacional
dom(TelComercial)=Número de telefone nacional
dom(MPA)=Média dos Pontos Acumulados

Uma relação r (ou instância de relação) da relação esquema $R(A_1, A_2, ..., A_n)$, também denotada por r(R), é um conjunto de tuplas $r=\{ t_1, t_2, ..., t_m\}$. Cada tupla t é uma lista ordenada de n valores $t=<v_1, v_2, ..., v_n>$, onde cada valor $v_i$, $1 \leq i \leq n$, é um elemento do $dom(A_i)$ ou um valor especial ***null***. São utilizados, com freqüência, os termos **intenção da relação** para o esquema R e **extensão da relação** para a instância r(R).

Atributos

| ESTUDANTE | Nome | NSS | Telefone | Endereço | TelComercial | Anos | MPA |
|---|---|---|---|---|---|---|---|
| | Joaquim | 305 | 555-444 | R. X, 123 | **null** | 19 | 3.21 |
| | Katarina | 381 | 555-333 | Av. K, 43 | **null** | 18 | 2.89 |
| tuplas | Daví | 422 | **null** | R. D, 12 | 555-678 | 25 | 3.53 |
| | Carlos | 489 | 555-376 | R. H, 9 | 555-789 | 28 | 3.93 |
| | Barbara | 533 | 555-999 | Av. f, 54 | **null** | 19 | 3.25 |

**Figura 5.2 – Exemplos de instâncias para uma relação ESTUDANTE.**

A Figura 5.2 mostra um exemplo de uma relação ESTUDANTE, que corresponde ao esquema estudante especificado anteriormente. Cada tupla na relação representa uma entidade estudante. A relação é mostrada em forma de tabela, onde cada tupla é representada pelas linhas e cada atributo na linha de cabeçalho indicando os papéis ou a interpretação dos valores encontrados em cada coluna.

### 5.1.1 Notação do Modelo Relacional

As seguintes notações serão utilizadas para apresentar alguns conceitos do modelo relacional:

- Uma relação esquema R de grau n é representada como $R(A_1, A_2, ..., A_n)$.
- Uma tupla t em uma relação r(R) é representada como $t=<v_1, v_2, ..., v_n>$, onde $v_i$ é o valor correspondente para atributos $A_i$. Serão utilizadas as seguintes notações para se referir aos valores dos componentes de tuplas:
  - $t[A_i]$ indica o valor de $v_i$ em t para o atributo $A_i$.
  - $t[A_u, A_w, ..., A_z]$ onde $A_u, A_w, ..., A_z$ é uma lista de atributos de R, indica o conjunto de valores $<v_u, v_w, ..., v_z>$ de t correspondentes aos atributos especificados na lista.
- As letras Q, R e S denotam nomes de relação.
- As letras q, r e s denotam instâncias de relação.
- As letras t, u e v denotam tuplas.
- Em geral, o nome de uma relação tal como ESTUDANTE indica o conjunto atual de tuplas na relação - instância corrente da relação - e ESTUDANTE(Nome, NSS, ...) refere-se à relação esquema.
- Os nomes de atributos são algumas vezes qualificados com o nome da relação a qual pertencem, por exemplo, ESTUDANTE.Nome ou ESTUDANTE.Anos.

Considere uma tupla t=<'Barbara', '533', '555-999', 'Av. f, 54', null, 19, 3.25> da relação ESTUDANTE da Figura 5.2; t[Nome]=<'Barbara'> e t[NSS, MPA, Anos]=<'533', 3.25, 19>.

### 5.1.2 Atributos-chaves de uma Relação

Uma relação é definida como um conjunto de tuplas. Pela definição, todos os elementos de um conjunto são distintos. Assim, todas as tuplas de uma relação também são distintas. Isto significa que nenhuma tupla pode ter a mesma combinação de valores para todos os seus atributos. Normalmente, existem **subconjuntos de atributos** de uma relação esquema R com a propriedade de que nenhuma tupla de uma relação r de R tenha a mesma combinação de valores para esses atributos. Suponha que esse subconjunto seja denotado por SC; então para quaisquer tuplas $t_1$ e $t_2$ em r de R, deve valer a regra:

$t_1[SC] \neq t_2[SC]$

Assim, SC é chamada **super-chave** da relação esquema R. Toda relação tem ao menos uma super-chave, que é o conjunto de todos os seus atributos. Uma **chave C**, de uma relação esquema R, é uma super-chave de R com a propriedade adicional de não se poder remover qualquer atributo A de C e continuar a ser super-chave de R. Assim, uma chave é uma super-chave mínima; uma super-chave da qual não se pode remover qualquer atributo.

Por exemplo, considere a relação ESTUDANTE da Figura 5.2. O conjunto de atributos {NSS} é uma super-chave de ESTUDANTE porque sabe-se que nenhum estudante irá ter o mesmo valor para NSS e também é uma chave, pois não se pode remover nenhum atributo. Qualquer conjunto de atributos que inclua NSS - por exemplo, {NSS, Nome, Anos} - será uma super-chave. No entanto, o conjunto {NSS, Nome, Anos} não é uma chave de ESTUDANTE porque removendo Nome ou Anos, ou ambos, o conjunto resultante será ainda uma super-chave.

O valor de um atributo-chave pode ser usado para identificar unicamente uma tupla em uma relação. Por exemplo, o valor 533, do atributo NSS, identifica unicamente a tupla correspondente à 'Barbara' na relação ESTUDANTE. Note-se, que a indicação de quais atributos que formam a chave deve ser feita na relação esquema, a fim de restringir qualquer duplicação de tuplas em quaisquer instâncias do esquema. A chave deve ser determinada pelo significado dos atributos na relação esquema e deve ser *invariante ao tempo*. Por exemplo, o atributo Nome da relação ESTUDANTE não pode ser indicada como chave, uma vez que nada garante a não ocorrência de homônimos.

Em geral, uma relação esquema pode ter mais que uma chave. Nestes casos, cada chave é chamada **chave-candidata**. Por exemplo, o esquema da relação ESTUDANTE poderia ter um atributo adicional Código, para indicar o código interno de estudantes na escola. Assim, o esquema teria duas chaves candidatas: NSS e Código. É comum designar uma das chaves candidatas como a **chave-primária** da relação. A indicação no modelo de qual chave-candidata é a chave-primária é realizada sublinhado-se os atributos que formam a chave-candidata escolhida. Quando uma relação esquema tem muitas chaves-candidatas, a escolha da chave-primária é arbitrária; no entanto, é sempre melhor escolher a chave-primária com o menor número de atributos.

### 5.1.3 Esquemas de Bases de Dados Relacionais e Restrições de Integridade

- Definição de um esquema de base de dados relacional e instância da base de dados relacional: um **esquema da base de dados relacional**, S, é um conjunto de relações esquemas $S=\{R_1, R_2, ..., R_m\}$ e um conjunto de restrições de integridade (RI). Uma **instância da base de dados relacional**, DB de S, é um conjunto de instâncias de relações $DB=\{r_1, r_2, ..., r_m\}$ tal que $r_i$ é uma instância de $R_i$ e que satisfaz as restrições de integridade especificadas em RI. A Figura 5.3 ilustra um esquema da base de dados relacional denominada COMPANHIA. O termo **base de dados relacional** refere-se, implicitamente, ao esquema e às suas instâncias.

**EMPREGADO**

| PNOME | MNOME | SNOME | NSS | DATANASC | ENDEREÇO | SEXO | SALARIO | NSSSUPER | NDEP |
|---|---|---|---|---|---|---|---|---|---|

**DEPARTAMENTO**

| DNOME | DNÚMERO | NSSGER | DATINICGER |
|---|---|---|---|

**LOCAIS_DEPTO**

| DNÚMERO | DLOCALIZAÇÃO |
|---|---|

**PROJETO**

| PNOME | PNÚMERO | PLOCALIZAÇÃO | DNUM |
|---|---|---|---|

**TRABALHA_EM**

| NSSEMP | PNRO | HORAS |
|---|---|---|

**DEPENDENTE**

| NSSEMP | NOMEDEPENDENTE | SEXO | DATANIV | RELAÇÃO |
|---|---|---|---|---|

**Figura 5.3 – Esquema da base de dados relacional COMPANHIA**

Observa-se, na Figura 5.3 acima, que o atributo DNÚMERO tanto de DEPARTAMENTO quanto de LOCAIS_DEPTO referem-se ao mesmo conceito do mundo real - o número dado a um departamento. Este mesmo conceito é chamado NDEP em EMPREGADO e DNUM em PROJETO. Isto significa que é permitido dar nomes de atributos distintos para um mesmo conceito do mundo real. Permite-se, também, que atributos que representam conceitos diferentes tenham o mesmo nome desde que em relações diferentes. Por exemplo, poderia ter sido usado NOME ao invés de PNOME e DNOME nas relações esquemas PROJETO e DEPARTAMENTO, respectivamente.

- Restrições de Integridade sobre um Esquema de Base de Dados Relacional: as **restrições de chave** especificam as chaves-candidatas de cada relação esquema; os valores das chaves-candidatas devem ser únicos para todas as tuplas de quaisquer instâncias da relação esquema. Além da restrição de chave, dois outros tipos de restrições são consideradas no modelo relacional: integridade de entidade e integridade referencial.

A **restrição de integridade de entidade** estabelece que nenhum valor da chave-primária pode ser nulo. Isso porque, o valor de uma chave-primária é utilizado para identificar tuplas em uma relação. Por exemplo, se duas ou mais tuplas tiverem o valor *null* para a chave-primária, não haverá como diferenciar uma tupla da outra.

As restrições de chave e de integridade de entidade aplicam-se apenas a relações individuais. A **restrição de integridade referencial** é uma restrição que é especificada entre duas relações e é usada para manter a consistência entre tuplas de duas relações. Informalmente, a restrição de integridade referencial estabelece que uma tupla de uma relação que se refere à outra relação deve se referir a uma tupla existente naquela relação. Por exemplo, na Figura 5.4, o atributo NDEP de EMPREGADO indica o número do departamento que cada empregado trabalha. Assim, todos os valores de NDEP nas tuplas da relação EMPREGADO devem pertencer ao conjunto de valores do atributo DNÚMERO da relação DEPARTAMENTO.

**EMPREGADO**

| PNOME | MNOME | SNOME | NSS | DATANASC | ENDEREÇO | SEXO | SALARIO | NSSSUPER | NDEP |
|---|---|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | 09-JAN-55 | R. A, 1 | M | 3000 | 333445555 | 5 |
| Franklin | T | Wong | 333445555 | 08-DEZ-45 | R. B, 2 | M | 4000 | 888665555 | 5 |
| Alicia | J | Zelaya | 999887777 | 19-JUL-58 | Av. C, 3 | F | 2500 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | 20-JUN-31 | Trav. D, 4 | F | 4300 | 888665555 | 4 |
| Ramesh | K | Narayan | 666884444 | 15-SET-52 | R. E, 5 | M | 3800 | 333445555 | 5 |
| Joyce | A | English | 453453453 | 31-JUL-62 | R. F, 6 | F | 2500 | 333445555 | 5 |
| Ahmad | V | Jabbar | 987987987 | 29-MAR-59 | Av G, 7 | M | 2500 | 987654321 | 4 |
| James | E | Borg | 888665555 | 10-NOV-27 | Av H, 8 | M | 5500 | null | 1 |

**DEPARTAMENTO**

| DNOME | DNÚMERO | NSSGER | DATINICGER |
|---|---|---|---|
| Pesquisa | 5 | 333445555 | 22-MAI-78 |
| Administrativo | 4 | 987654321 | 01-JAN-85 |
| Gerencial | 1 | 888665555 | 19-JUN-71 |

**LOCAIS_DEPTO**

| DNÚMERO | DLOCALIZAÇÃO |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugariand |
| 5 | Houston |

**PROJETO**

| PNOME | PNÚMERO | PLOCALIZAÇÃO | DNUM |
|---|---|---|---|
| ProdutoX | 1 | Bellaire | 5 |
| ProdutoY | 2 | Sugarland | 5 |
| ProdutoZ | 3 | Houston | 5 |
| Automação | 10 | Stafford | 4 |
| Reorganização | 20 | Houston | 1 |
| Beneficiamento | 30 | Stafford | 4 |

**TRABALHA_EM**

| NSSEMP | PNRO | HORAS |
|---|---|---|
| 123456789 | 1 | 32.5 |
| 123456789 | 2 | 7.5 |
| 666884444 | 3 | 40.0 |
| 453453453 | 1 | 20.0 |
| 453453453 | 2 | 20.0 |
| 333445555 | 2 | 10.0 |
| 333445555 | 3 | 10.0 |
| 333445555 | 10 | 10.0 |
| 333445555 | 20 | 10.0 |
| 999887777 | 30 | 30.0 |
| 999887777 | 10 | 10.0 |
| 987987987 | 10 | 35.0 |
| 987987987 | 30 | 5.0 |
| 987654321 | 30 | 20.0 |
| 987654321 | 20 | Null |

**DEPENDENTE**

| NSSEMP | NOMEDEPENDENTE | SEXO | DATANIV | RELAÇÃO |
|---|---|---|---|---|
| 333445555 | Alice | F | 05-ABR-76 | FILHA |
| 333445555 | Theodore | M | 25-OUT-73 | FILHO |
| 333445555 | Joy | F | 03-MAI-48 | ESPOSA |
| 987654321 | Abner | M | 29-FEV-78 | MARIDO |
| 123456789 | Michael | M | 01-JAN-78 | FILHO |
| 123456789 | Alice | F | 31-DEZ-78 | FILHA |
| 123456789 | Elizabeth | F | 05-MAI-57 | ESPOSA |

**Figura 5.4 – Instâncias para a base de dados relacional COMPANHIA**

Para definir formalmente a restrição de integridade referencial, há a necessidade de antes definir o conceito de **chave-estrangeira** (CE). As condições para uma chave-estrangeira, descritas abaixo, especificam uma restrição de integridade referencial entre duas relações esquemas $R_1$ e $R_2$. Um conjunto de atributos CE na relação esquema $R_1$ será uma **chave-estrangeira** de $R_1$ se ele satisfizer as seguintes regras:

1. Os atributos em CE têm o mesmo domínio dos atributos da chave-primária CP da outra relação esquema $R_2$. Diz-se que os atributos CE **referenciam** ou **referem-se** à relação $R_2$.
2. Uma CE na tupla $t_1$ ou tem um valor que ocorre como CP de alguma tupla $t_2$ de $R_2$ ou tem o valor *null*. No primeiro caso, tem-se $t_1[CE]=t_2[CP]$, e diz-se que $t_1$ **referencia** ou **refere-se** à tupla $t_2$.

Uma base de dados tem muitas relações, e usualmente possuem muitas restrições de integridade referencial. Para especificar estas restrições, o projetista deve ter um claro entendimento do significado ou papel que os atributos desempenham nas diversas relações esquemas da base de dados. Normalmente, as restrições de integridade referencial são derivadas dos relacionamentos entre entidades representadas pelas relações esquemas. Por exemplo, considere a base de dados mostrada na 5.4. Na relação EMPREGADO, o atributo NDEP refere-se ao departamento em que cada empregado trabalha; desse modo, designa-se NDEP como a chave-estrangeira de EMPREGADO referenciando a relação DEPARTAMENTO. Isso significa que um valor de NDEP em alguma tupla $t_1$ da relação EMPREGADO deve ter um valor correspondente para a chave-primária da relação DEPARTAMENTO - o atributo DNÚMERO - em alguma tupla $t_2$ da relação DEPARTAMENTO ou o valor de NDEP pode ser *null* se o empregado não pertencer a nenhum departamento. Na Figura 5.4, a tupla do empregado "John Smith" referencia a tupla departamento de "Pesquisa", indicando que "John Smith" trabalha para este departamento.

Note-se que uma chave-estrangeira pode referenciar sua própria relação. Por exemplo, o atributo NSSSUPER em EMPREGADO refere-se ao supervisor de um empregado; isto é, um outro empregado. Pode-se, diagramaticamente, mostrar as restrições de integridade desenhando-se arcos direcionados partindo da chave-estrangeira para a relação referenciada. A Figura 5.5 ilustra o esquema apresentado na Figura 5.3 com as restrições de integridade referencial anotadas desta maneira.

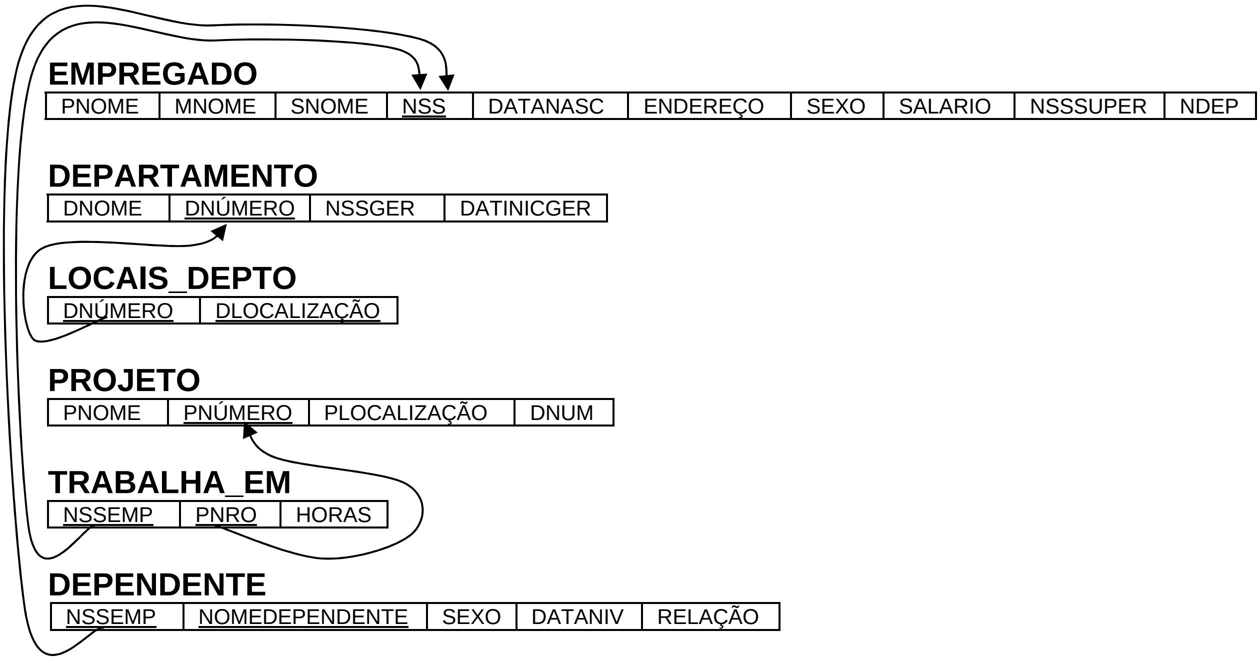



**Figura 5.5 - Esquema COMPANHIA com restrições de integridade**

Todas as restrições de integridade deveriam ser especificadas no esquema da base de dados relacional se o projetista tiver interesse em manter essas restrições válidas para toda a base de dados. Assim, em um sistema relacional, a linguagem de definição de dados (DDL) deveria fornecer recursos para especificar os vários tipos de restrições tal que o SGDB possa verificá-las automaticamente. Muitos sistemas de gerenciamento de base de dados relacionais permitem restrições de chave e de integridade de entidade mas, infelizmente, alguns não permitem a integridade referencial.

Além dos tipos de restrições descritas acima, existem outras mais gerais, algumas vezes chamadas de *restrições de integridade semânticas*, que podem ser especificadas e verificadas numa base de dados relacional. Exemplos de tais restrições são "o salário de um empregado não deve ultrapassar o salário de seu supervisor" e "o número máximo de horas por semana que um empregado pode trabalhar num projeto é 56". Tais restrições não são verificadas por SGBD relacionais atualmente encontrados no mercado.

### 5.1.4 Operações de Atualizações sobre Relações

Existem três tipos básicos de operação de atualização sobre relações - inserção, remoção e modificação. A **inserção** é usada para inserir novas tuplas em uma relação, a **remoção** elimina tuplas e a **modificação** modifica os valores de alguns atributos. Quando são aplicadas operações de atualização, o projetista deve verificar que as restrições de integridade especificadas no esquema da base de dados relacional não sejam violadas.

# 6 Mapeamento do MER para o Modelo de Dados Relacional

É comum, em projetos de banco de dados realizarem a modelagem dos dados através de um modelo de dados de alto-nível. Os produtos gerados por esse processo são os esquemas de visões que são posteriormente integradas para formar um único esquema. O modelo de dados de alto-nível normalmente adotado é o Modelo Entidade-Relacionamento (MER) e o esquema das visões e de toda a base de dados são especificados em diagramas entidade-relacionamento (DER).

O passo seguinte à modelagem dos dados é o mapeamento do diagrama da base de dados global, obtido na fase anterior, para um modelo de dados de implementação. Existem três tipos de modelos de dados de implementação: hierárquico, rede e relacional. Para cada um desses modelos, pode-se definir estratégias de tradução a partir de um DER específico. A estratégia de tradução, ou de mapeamento, tratada neste capítulo é para o modelo de dados relacional.

Para isso, considere o esquema relacional mostrado na Figura 6.1 (semelhante à Figura 5.5) que foi derivado do DER da Figura 4.11 seguindo um procedimento de mapeamento. Este procedimento é apresentado passo-a-passo, a partir do exemplo do DER COMPANHIA.

**EMPREGADO**

| | | | | | | | | ce | ce |
|---|---|---|---|---|---|---|---|---|---|
| PNOME | MNOME | SNOME | <u>NSS</u> | DATANASC | ENDEREÇO | SEXO | SALARIO | NSSSUPER | DNUM |
| | | | cp | | | | | | |

**DEPARTAMENTO**

| | | ce | * |
|---|---|---|---|
| DNOME | <u>DNÚMERO</u> | NSSGER | DATINICGER |
| | cp | | |

**LOCAIS_DEPTO**

Ce

| <u>DNÚMERO</u> | <u>DLOCALIZAÇÃO</u> |
|---|---|

_______________________/
Cp

**PROJETO**

| | | | ce |
|---|---|---|---|
| PNOME | <u>PNÚMERO</u> | PLOCALIZAÇÃO | DNUM |
| | cp | | |

**TRABALHA_EM**

| Ce | ce | |
|---|---|---|
| <u>NSSEMP</u> | <u>PNRO</u> | HORAS* |

______________/
Cp

**DEPENDENTE**

Ce

| <u>NSSEMP</u> | <u>NOMEDEPENDENTE</u> | SEXO | DATANIV | RELAÇÃO |
|---|---|---|---|---|

__________________________/
Cp

**Figura 6.1 – Modelo Relacional para o esquema COMPANHIA.**

Passo 1: Para cada entidade regular E no DER, criar uma relação R que inclua todos os atributos simples de E. Para um atributo composto, inclua apenas os atributos simples que compõem o atributo composto. Escolha um dos atributos-chave de E como sendo a chave-primária de R. Se a chave escolhida de E for composta, então o conjunto de atributos simples que o compõem irão formar a chave-primária de R.

No exemplo, foram criadas as relações EMPREGADO, DEPARTAMENTO e PROJETO correspondentes às entidades regulares EMPREGADO, DEPARTAMENTO e PROJETO presentes no DER. Os atributos indicados com ce (chave-estrangeira) ou * (atributos de relacionamento) não foram incluídos ainda; eles serão adicionados durante os passos subseqüentes. Foram escolhidas as chaves-primárias NSS, DNÚMERO e PNÚMERO para as relações EMPREGADO, DEPARTAMENTO e PROJETO respectivamente.

Passo 2: Para cada tipo de entidade fraca W do DER com o tipo de entidade de identificação E, criar uma relação R e incluir todos os atributos simples (ou os componentes simples de atributos compostos) de W como atributos de R. Além disso, incluir como a chave-estrangeira de R a chave-primária da relação que corresponde ao tipo de entidade de identificação; isto resolve o problema do tipo do relacionamento de identificação de W. A chave-primária de R é a combinação da chave-primária do tipo de entidade de identificação e a chave-parcial do tipo de entidade fraca W.

No exemplo, foi criada a relação DEPENDENTE correspondente ao tipo de entidade fraca DEPENDENTE do DER. Foi incluída a chave-primária da relação EMPREGADO - que corresponde ao tipo de entidade de identificação - como um atributo de DEPENDENTE; foi renomeado o atributo NSS para NSSEMP, embora não seja necessário. A chave-primária da relação DEPENDENTE é a combinação {NSSEMP, NOMEDEPENDENTE} porque NOMEDEPENDENTE é chave-parcial de DEPENDENTE.

Passo 3: Para cada tipo de relacionamento binário 1:1 R do DER, criar as relações S e T que correspondem aos tipos de entidade participantes em R. Escolher uma das relações, por exemplo S, que inclua como chave-estrangeira de S a chave-primária de T. É melhor escolher o tipo de entidade com participação total em R como a relação S. Inclua todos os atributos simples (ou os componentes simples de atributos compostos) do tipo de relacionamento 1:1 R como atributos de S.

No exemplo, foi mapeado o tipo de relacionamento 1:1 GERENCIA da Figura 4.10, escolhendo o tipo de entidade participante DEPARTAMENTO para fazer o papel de S porque sua participação no tipo de relacionamento GERENCIA é total (todo departamento tem um gerente). Foi incluída a chave-primária da relação EMPREGADO como a chave-estrangeira na relação DEPARTAMENTO que foi chamado de NSSGER . Também foi incluído o atributo simples DataInício do tipo de relacionamento GERENCIA na relação DEPARTAMENTO e foi renomeado como DATINICGER.

Note-se que, uma alternativa para o mapeamento de um tipo de relacionamento 1:1 seria unir os dois tipos de entidades e o tipo de relacionamento numa única relação. Isto é particularmente apropriado quando ambas as participações são total e quando os tipos de entidade não participam em quaisquer outros tipos de relacionamentos.

Passo 4: Para cada tipo de relacionamento binário regular 1:N (não fraca) R, identificar a relação S que representa o tipo de entidade que participa do lado N do tipo de relacionamento. Inclua como chave-estrangeira de S a chave-primária da relação T que representa o outro tipo de entidade que participa em R; isto porque cada instância da entidade do lado 1 está relacionada a mais de uma instância de entidade no lado N do tipo de relacionamento. Por exemplo, no tipo de relacionamento 1:N TRABALHA-PARA cada empregado está relacionado a um único departamento. Inclua também quaisquer atributos simples (ou componentes simples de atributos compostos) do tipo de relacionamento 1:N como atributos de S.

No exemplo, foram mapeados os tipos de relacionamentos 1:N TRABALHA-PARA e SUPERVISIONA. Para TRABALHA-PARA incluiu-se a chave-primária da relação DEPARTAMENTO como a chave-estrangeira na relação EMPREGADO e foi chamado DNUM. Para SUPERVISIONA incluiu-se a chave-primária da relação EMPREGADO como a chave-estrangeira na relação EMPREGADO e foi denominado NSSSUPER. O relacionamento CONTROLA é mapeado da mesma maneira.

Passo 5: Para cada tipo de relacionamento binário M:N R, criar uma nova relação S para representar R. Incluir como chave-estrangeira em S as chaves-primárias das relações que representam os tipos de entidade participantes; sua combinação irá formar a chave-primária de S. Inclua também qualquer atributo simples do tipo de relacionamento M:N (ou componentes simples dos atributos compostos) como atributos de S. Note-se que não se pode representar um tipo de relacionamento M:N como uma simples chave-estrangeira em uma das relações participantes - como foi feito para os tipos de relacionamentos 1:1 e 1:N - por causa da razão de cardinalidade M:N. Relacionamentos M:N sempre derivam uma nova relação, para o tipo relacionamento.

No exemplo foi mapeado o tipo de relacionamento M:N TRABALHA-EM criando-se a relação TRABALHA_EM na Figura 4.6. Incluiu-se as chaves-primárias das relações PROJETO e EMPREGADO como chaves-estrangeiras em TRABALHA_EM e foi renomeado NSS para NSSEMP e Número para PNRO respectivamente. Também foi incluído um atributo HORAS para representar o atributo Horas do tipo de relacionamento. A chave-primária da relação TRABALHA_EM é a combinação das chaves-estrangeiras {NSSEMP, PNRO}.

Note-se que sempre é possível mapear relacionamentos 1:1 ou 1:N da mesma maneira que os relacionamentos M:N. Esta alternativa pode ser adotada desde que existam poucas instâncias de relacionamento a fim de evitar valores *null* em chaves-estrangeiras, ou em casos em que cardinalidade irá ser futuramente modificada de 1:1 ou 1:N para M:N.

Passo 6: Para cada atributo A multivalorado, criar uma nova relação R que inclua um atributo correspondendo a A e a chave-primária K da relação que representa o tipo de entidade ou o tipo de relacionamento que tem A como atributo. A chave-primária de R é a combinação de A e K. Se o atributo multivalorado é composto inclua os atributos simples que o compõem.

No exemplo foi criada a relação LOCAIS_DEPTO. O atributo DLOCALIZAÇÃO representa o atributo multivalorado Localização de DEPARTAMENTO, enquanto DNÚMERO - como chave-estrangeira - representa a chave-primária da relação DEPARTAMENTO. A chave-primária de LOCAIS_DEPTO é a combinação de {DNÚMERO, DLOCALIZAÇÃO}. Uma tupla irá existir em LOCAIS_DEPTO para cada localização que um departamento tiver.

A Figura 6.1 mostra o esquema da base de dados relacional obtida aplicando-se os passos acima, e a Figura 5.4 mostra uma instância deste esquema. Observa-se que não foi discutido o mapeamento de tipos de relacionamento n-ário (n > 2) porque não existem na Figura 4.10. Estes tipos de relacionamentos podem ser mapeados da mesma forma que os tipos de relacionamentos M:N, apenas observando o seguinte passo adicional no procedimento de mapeamento.

Passo 7: Para cada tipo de relacionamento n-ário R, n>2, criar uma nova relação S para representar R. Inclua como chave-estrangeira em S as chaves-primárias das relações que representam os tipos de entidades participantes. Incluindo-se também qualquer atributo simples do tipo de relacionamento n-ário (ou componentes simples dos atributos compostos) como atributo de S. A chave-primária de S é normalmente uma combinação de todas as chaves-estrangeiras e referencia as relações que representam os tipos de entidades participantes. Porém, se a restrição de

participação (min, max) de um dos tipos de entidades E que participa em R tiver max=1, então a chave-primária de S pode ser a chave-estrangeira que referencia a relação E' correspondente a E; isto porque cada entidade e em E irá participar em apenas uma instância de R e, portanto, pode identificar univocamente esta instância de relacionamento.

Por exemplo, considere o tipo de relacionamento da Figura 4.13a. Este relacionamento triplo pode ser mapeado para a relação FORNECE mostrada na Figura 6.2, onde a chave-primária é a combinação das chaves-estrangeiras {FNOME, PNOME, NÚMERO}.

FORNECEDOR

| FNOME | ... |
|---|---|

PROJETO

| PNOME | ... |
|---|---|

PEÇA

| NÚMERO | ... |
|---|---|

FORNECE

| FNOME | PNOME | NÚMERO | **QUANTIDADE** |
|---|---|---|---|

**Figura 6.2 – Exemplo de mapeamento de um relacionamento ternário.**

O principal ponto que deve ser considerado em um esquema relacional, quando comparado ao esquema do MER, é que os tipos de relacionamento não são representados explicitamente; eles são representados por dois atributos A e B, um para a chave-primária e outra para a chave-estrangeira – sobre o mesmo domínio – incluídos em duas relações S e T. Duas tuplas em S e T estão relacionadas quando elas tiverem o mesmo valor para A e B, ou seja, os relacionamentos são definidos pelos valores dos atributos A e B.

# 7 Linguagens Formais de Consulta

## 7.1 Álgebra Relacional

As discussões anteriores sobre o modelo relacional contemplaram apenas os aspectos estruturais. Agora, a atenção será voltada para a álgebra relacional, que é uma coleção de operações utilizadas para manipular relações. Essas operações são usadas para selecionar tuplas de uma determinada relação ou para combinar tuplas relacionadas a diversas relações com o propósito de especificar uma consulta - uma requisição de recuperação - sobre a base de dados.

As operações da álgebra relacional são normalmente divididas em dois grupos. O primeiro deles inclui um conjunto de operações da teoria de conjuntos. As operações são UNION, INTERSECTION, DIFFERENCE e CARTESIAN PRODUCT. O segundo grupo consiste de operações desenvolvidas especificamente para bases de dados relacionais, tais como: SELECT, PROJECT e JOIN entre outras.

### 7.1.1 Operações SELECT e PROJECT

#### 7.1.1.1 O Operador SELECT

A operação SELECT é usada para selecionar um subconjunto de tuplas de uma relação as quais devem satisfazer uma **condição de seleção**. Por exemplo, a seleção de um subconjunto de tuplas da relação EMPREGADOS que trabalham para o departamento 4 ou que tenham salário maior que 3000. Cada uma dessas condições é especificada individualmente usando a operação SELECT como segue:

$$\sigma_{\text{NDEP = 4}}\ (\text{EMPREGADO})$$
$$\sigma_{\text{SALÁRIO > 3000}}\ (\text{EMPREGADO})$$

Em geral, a operação SELECT é denotada por:

$$\sigma_{\text{<condição de seleção>}}\ (\text{<nome da relação>})$$

onde o símbolo $\sigma$ é usado para denotar o operador SELECT e a condição de seleção é uma expressão Booleana especificada sobre atributos da relação especificada.

A relação resultante da operação SELECT tem os mesmos atributos da relação especificada em <nome da relação>. A expressão booleana especificada em <condição de seleção> é construída a partir de cláusulas da forma:

<nome de atributo> <operador de comparação> <valor constante>, ou
<nome de atributo> <operador de comparação> <nome de atributo>

Onde <nome de atributo> é o nome de um atributo da <nome da relação>, <operador de comparação> é normalmente um dos operadores relacionais {=,<,>, ...} <valor constante> é um valor constante. As cláusulas podem ser utilizadas em conjunto com os operadores lógicos {AND, OR NOT} para formar uma condição de seleção composta. Por exemplo, suponha que se deseja selecionar as tuplas de todos os empregados que ou trabalham no departamento 4

com salário superior a R$2.500,00 ou trabalham no departamento 5 e ganham mais que R$3.000,00. Neste caso, pode-se especificar a consulta da seguinte forma:

$\sigma_{\text{(NDEP = 4 AND SALÁRIO > 2500) OR (NDEP = 5 AND SALÁRIO > 3000)}}\text{(EMPREGADO)}$

O resultado é mostrado na Figura 7.1a.

(a)

| PNOME | MNOME | SNOME | NSS | DATANASC | ENDEREÇO | SEXO | SALARIO | NSSSUPER | NDEP |
|---|---|---|---|---|---|---|---|---|---|
| Franklin | T | Wong | 333445555 | 08-DEZ-45 | R. B, 2 | M | 4000 | 888665555 | 5 |
| Jennifer | S | Wallace | 987654321 | 20-JUN-31 | Trav. D, 4 | F | 4300 | 888665555 | 4 |
| Ramesh | K | Narayan | 666884444 | 15-SET-52 | R. E, 5 | M | 3800 | 333445555 | 5 |

(b)

| SNOME | PNOME | SALARIO |
|---|---|---|
| Smith | John | 3000 |
| Wong | Franklin | 4000 |
| Zelaya | Alícia | 2500 |
| Wallace | Jennifer | 4300 |
| Narayan | Ramesh | 3800 |
| English | Joyce | 2500 |
| Jabbar | Ahmad | 2500 |
| Borg | James | 5500 |

(c)

| SEXO | SALARIO |
|---|---|
| M | 3000 |
| M | 4000 |
| F | 2500 |
| F | 4300 |
| M | 3800 |
| M | 2500 |
| M | 5500 |

**Figura 7.1 – Exemplo de utilização do operador SELECT**

O operador SELECT é unário; isto é, ele é aplicado somente a uma relação. Assim, o SELECT não pode ser usado para selecionar tuplas de mais de uma relação. Observe também que a operação de seleção é aplicada individualmente para cada tupla. Assim, as condições de seleção não podem ser aplicadas a mais que uma tupla. O grau da relação resultante é a mesma que a relação original. O número de tuplas da relação resultante é sempre menor ou igual ao número de tuplas da relação original.

Note que o operador SELECT é **comutativo**; isto é,
$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$

Assim, uma seqüência de SELECTs pode ser aplicada em qualquer ordem. Além disso, pode-se sempre trocar operadores SELECT em cascata com a conjuntiva AND; isto é:

$\sigma_{<cond1>}(\sigma_{<cond2>}(...(\sigma_{<condn>}(R))...)) = \sigma_{<cond1>\text{ AND }<cond2>\text{ AND ... AND }<condn>}(R)$

### 7.1.1.2 O Operador PROJECT

Pensando na relação como uma tabela, o operador SELECT seleciona algumas linhas da tabela enquanto descarta outras. O operador PROJECT, por outro lado, seleciona certas colunas da tabela e descarta outras. Se existir o interesse sobre certos atributos da relação, pode-se usar o PROJECT para "projetar" a relação sobre esses atributos. Por exemplo, suponha a necessidade de listar, para cada empregado, os atributos PNOME, SNOME e SALÁRIO; então pode-se usar o PROJECT como segue:

$\pi_{\text{SNOME, PNOME, SALÁRIO}}\text{(EMPREGADO)}$

A relação resultante é mostrada na Figura 7.1b. A forma geral do operador PROJECT é:

$\pi_{\text{<lista de atributos>}}(\text{<nome da relação>})$

onde $\pi$ é o símbolo usado para representar o operador PROJECT e <lista de atributos> é uma lista de atributos da relação especificada por <nome da relação>. A relação resultante tem apenas os atributos especificados em <lista de atributos> e estes atributos aparecem na mesma ordem em que foram especificados. Assim, o grau é igual ao número de atributos em <lista de atributos>.

Convém salientar que, caso a lista de atributos não contenha atributos chaves, então é provável que tuplas duplicadas apareçam no resultado. A operação PROJECT remove implicitamente quaisquer tuplas duplicadas, tal que o resultado da operação PROJECT seja um conjunto de tuplas e assim, uma relação válida. Por exemplo, considere a seguinte operação:

$\pi_{\text{SEXO, SALÁRIO}}$ (EMPREGADO)

O resultado é mostrado na Figura 7.1c, onde a tupla <F, 2500> aparece apenas uma vez, mesmo que esta combinação de valores apareça duas vezes na relação EMPREGADO. Dessa maneira, se duas ou mais tuplas idênticas aparecerem quando aplicada a operação PROJECT, apenas uma será mantida no resultado; isto é conhecido como **eliminação de duplicidade** e é necessário para assegurar que o resultado da operação também seja uma relação - um conjunto de tuplas.

O número de tuplas na relação resultante sempre será igual ou menor que a quantidade de tuplas na relação original.

Note-se que:

$\pi_{\text{<lista1>}} (\pi_{\text{<lista2>}} (R)) = \pi_{\text{<lista1>}} (R)$

caso <lista2> contenha os atributos de <lista1>; caso contrário, o lado esquerdo da igualdade acima estará incorreto. A comutatividade não é válida para PROJECT.

### 7.1.2 Seqüência de Operações

As relações mostradas na Figura 7.1 não possuem nomes. Em geral, existe a necessidade de se aplicar várias operações da álgebra relacional uma após a outra. Pode-se escrever essas operações em apenas uma única expressão da álgebra relacional ou aplicar uma única operação por vez e criar relações intermediárias. Neste último caso, deve-se dar nomes às relações intermediárias. Por exemplo, deseja-se recuperar o primeiro nome, o último nome e o salário de todos os empregados que trabalham no departamento 5. Isto pode ser feito aplicando-se os operadores SELECT e PROJECT:

$\pi_{\text{PNOME, SNOME, SALÁRIO}} (\sigma_{\text{NDEP=5}}$ (EMPREGADO))

A Figura 7.2a mostra o resultado desta expressão da álgebra relacional. Alternativamente, pode-se explicitar a seqüência de operações, dando um nome para cada relação intermediária:

DEP5_EMPS $\leftarrow \sigma_{\text{NDEP=5}}$ (EMPREGADO)
RESULT $\leftarrow \pi_{\text{PNOME, SNOME, SALÁRIO}}$ (DEP5_EMPS)

(a)

| PNOME | SNOME | SALÁRIO |
|---|---|---|
| John | Smith | 3000 |
| Franklin | Wong | 4000 |
| Alícia | Zelaya | 2500 |
| Jennifer | Wallace | 4300 |
| Ramesh | Narayan | 3800 |
| Joyce | English | 2500 |
| Ahmad | Jabbar | 2500 |
| James | Borg | 5500 |

DEP5_EMPS
(b)

| PNOME | MNOME | SNOME | NSS | DATANASC | ENDEREÇO | SEXO | SALÁRIO | NSSSUPER | NDEP |
|---|---|---|---|---|---|---|---|---|---|
| John | B | Smith | 123456789 | 09-JAN-55 | R. A, 1 | M | 3000 | 333445555 | 5 |
| Franklin | T | Wong | 333445555 | 08-DEZ-45 | R. B, 2 | M | 4000 | 888665555 | 5 |
| Ramesh | K | Narayan | 666884444 | 15-SET-52 | R. E, 5 | M | 3800 | 333445555 | 5 |
| Joyce | A | English | 453453453 | 31-JUL-62 | R. F, 6 | F | 2500 | 333445555 | 5 |

RESULT

| PRIMEIRO_NOME | SOBRENOME | SALÁRIO |
|---|---|---|
| John | Smith | 3000 |
| Franklin | Wong | 4000 |
| Ramesh | Narayan | 3800 |
| Joyce | English | 2500 |

**Figura 7.2 – Exemplo de execução de uma expressão da Álgebra Relacional**

## 7.1.3 Renomeando Atributos

Normalmente, é mais simples dividir uma seqüência de operações especificando as relações intermediárias ao invés de escrever uma única expressão da álgebra relacional. Pode-se, também, utilizar a técnica de **renomear atributos** nas relações intermediárias. Para renomear atributos de uma relação, que resultou da aplicação de uma operação da álgebra relacional, basta listar os nomes dos atributos entre parênteses:

DEP5_EMPS $\leftarrow \sigma_{NDEP=5}$ (EMPREGADO)

RESULT(PRIMEIRO_NOME, SOBRENOME, SALÁRIO)$\leftarrow \pi_{PNOME, SNOME, SALÁRIO}$ (DEP5_EMPS)

Os resultados das duas operações acima são ilustrados na Figura 7.2b.

## 7.1.4 Operações da Teoria dos Conjuntos

O próximo grupo de operações da álgebra relacional são as operações matemáticas padrões sobre conjuntos. Elas se aplicam ao modelo relacional porque uma relação é definida como um conjunto de tuplas. Por exemplo, suponha a necessidade de se recuperar o número do seguro social de todos os empregados que trabalham no departamento 5 ou, diretamente supervisione um empregado que trabalha no departamento 5. Esta operação pode ser realizada usando o operador UNION.

DEP5_EMPS $\leftarrow \sigma_{NDEP=5}$ (EMPREGADO)

RESULT1$\leftarrow \pi_{NSS}$ (DEP5_EMPS)

RESULT2(NSS)$\leftarrow \pi_{NSSSUPER}$ (DEP5_EMPS)

RESULT$\leftarrow$RESULT1 $\cup$ RESULT2

A relação RESULT1 contém o número do seguro social de todos os empregados que trabalham no departamento 5. RESULT2 contém o número do seguro social de todos os empregados que diretamente supervisionam empregados que trabalham no departamento 5. A

operação UNION gera uma relação que contém tanto as tuplas de RESULT1 quanto de RESULT2. A Figura 7.3 ilustra este exemplo.

| RESULT1 | NSS |
|---|---|
| | 123456789 |
| | 333445555 |
| | 666884444 |
| | 453453453 |

| RESULT2 | NSS |
|---|---|
| | 333445555 |
| | 888665555 |

| RESULT | NSS |
|---|---|
| | 123456789 |
| | 333445555 |
| | 666884444 |
| | 453453453 |
| | 888665555 |

**Figura 7.3 – Exemplo de aplicação do operador UNION.**

Existem várias operações da teoria de conjuntos que são utilizadas para agrupar elementos de dois conjuntos, entre elas estão: UNION, INTERSECTION e DIFFERENCE. Estas operações são binárias; isto é, elas necessitam de dois conjuntos. Quando essas operações são adaptadas para a base de dados relacional, deve-se assegurar que essas operações resultem sempre em relações válidas. Para conseguir isso, as duas relações aplicadas a qualquer uma das três operações acima devem ter o mesmo **tipo de tuplas**; esta condição é chamada união compatível. Duas relações $R(A_1, A_2, ..., A_n)$ e $S(B_1, B_2, ..., B_n)$ são **união compatível** se elas tiverem o mesmo grau n, e $dom(A_i)=dom(B_i)$ para $1 \leq i \leq n$. Isso significa que as duas relações têm o mesmo número de atributos e que cada par de atributos correspondentes tem o mesmo domínio.

Pode-se definir as três operações UNION, INTERSECTION e DIFFERENCE sobre duas relações que sejam união compatível R e S:

- UNION — O resultado da operação, denotado por $R \cup S$, é uma relação que inclui todas as tuplas de R e todas as tuplas de S. Tuplas duplicadas são eliminadas.
- INTERSECTION — O resultado desta operação, denotado por $R \cap S$, é a relação que inclui todas as tuplas que são comuns a R e S.
- DIFFERENCE — O resultado desta operação, denotado por R - S, é a relação que inclui todas as tuplas de R, mas que não estão em S.

Note que as operações UNION e INTERSECTION são operações comutativas:

$$R \cup S = S \cup R, \text{ e } R \cap S = S \cap R.$$

Estas operações podem ser aplicadas a qualquer número de relações, e ambas são associativas:

$$R \cup (S \cup T) = (R \cup S) \cup T, \text{ e}$$
$$R \cap (S \cap T) = (R \cap S) \cap T.$$

A operação DIFFERENCE não é comutativa:

$$R - S \neq S - R.$$

A operação CARTESIAN PRODUCT, denotada por $\chi$, é também uma operação de conjunto binária, mas as relações sobre as quais são aplicadas não necessitam ser união compatível. Esta operação é usada para combinar tuplas de duas relações tal que tuplas relacionadas possam ser identificadas.

Em geral, o resultado de $R(A_1, A_2, ..., A_n) \chi S(B_1, B_2, ..., B_m)$ é a relação Q com n + m atributos $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ nesta ordem. A relação resultante Q tem uma tupla para cada combinação de tuplas. Assim, se R tem nR tuplas e S tem nS tuplas, então $R \chi S$ terá nR*nS tuplas. Para ilustrar, considere que se deseja recuperar para cada empregado do sexo feminino uma lista de nomes de seus dependentes; isso pode ser feito da seguinte forma:

$EMP_FEM \leftarrow \sigma_{SEXO='F'}(EMPREGADO)$

$EMP_NOMES \leftarrow \pi_{PNOME,\ SNOME,\ NSS}(EMP_FEM)$

$EMP_DEP \leftarrow EMP_NOMES\ \chi\ DEPENDENTE$

$DEP_ATUAL \leftarrow \sigma_{NSS=NSSEMP}(EMP_DEP)$

$RESULT \leftarrow \pi_{PNOME,\ SNOME,\ NOMEDEPENDENTE}(DEP_ATUAL)$

As tuplas geradas a partir da seqüência de operações acima são mostradas na Figura 7.4. A relação EMP_DEP é o resultado da operação CARTESIAN PRODUCT entre EMP_NOMES da Figura 7.4, e DEPENDENTE da Figura 5.4. Em EMP_DEP, cada tupla de EMP_NOMES é combinada com todas as tuplas de DEPENDENTE, gerando um resultado que não tem muito significado. Deseja-se apenas combinar tuplas de empregado feminino com seus dependentes, o que significa dizer: tuplas de DEPENDENTES onde os valores de NSSEMP são iguais aos valores de NSS de EMPREGADO. Em DEP_ATUAL, foi obtido isto.

O CARTESIAN PRODUCT cria tuplas com atributos combinados de duas relações. Pode-se então selecionar apenas as tuplas que estejam relacionadas especificando uma condição de seleção apropriada, como foi feita no exemplo. Devido à seqüência: CARTESIAN PRODUCT seguido de SELECT, ser muito comum para se identificar tuplas relacionadas de duas relações, uma operação especial JOIN foi criada para especificar esta seqüência como uma única operação. Assim, a operação CARTESIAN PRODUCT é raramente utilizada isoladamente.

**EMP_FEM**

| PNOME | MNOME | SNOME | NSS | DATANASC | ENDEREÇO | SEXO | SALARIO | NSSSUPER | NDEP |
|---|---|---|---|---|---|---|---|---|---|
| Alicia | J | Zelaya | 999887777 | 19-JUL-58 | Av. C, 3 | F | 2500 | 987654321 | 4 |
| Jennifer | S | Wallace | 987654321 | 20-JUN-31 | Trav. D, 4 | F | 4300 | 888665555 | 4 |
| Joyce | A | English | 453453453 | 31-JUL-62 | R. F, 6 | F | 2500 | 333445555 | 5 |

**EMP_NOMES**

| PNOME | SNOME | NSS |
|---|---|---|
| Alicia | Zelaya | 999887777 |
| Jennifer | Wallace | 987654321 |
| Joyce | English | 453453453 |

**EMP_DEP**

| PNOME | SNOME | NSS | NSSEMP | NOMEDEPENDENTE | SEXO | DATANIV | RELAÇÃO |
|---|---|---|---|---|---|---|---|
| Alicia | Zelaya | 999887777 | 333445555 | Alice | F | 05-ABR-76 | FILHA |
| Alicia | Zelaya | 999887777 | 333445555 | Theodore | M | 25-OUT-73 | FILHO |
| Alicia | Zelaya | 999887777 | 333445555 | Joy | F | 03-MAI-48 | ESPOSA |
| Alicia | Zelaya | 999887777 | 987654321 | Abner | M | 29-FEV-78 | MARIDO |
| Alicia | Zelaya | 999887777 | 123456789 | Michael | M | 01-JAN-78 | FILHO |
| Alicia | Zelaya | 999887777 | 123456789 | Alice | F | 31-DEZ-78 | FILHA |
| Alicia | Zelaya | 999887777 | 123456789 | Elizabeth | F | 05-MAI-57 | ESPOSA |
| Jennifer | Wallace | 987654321 | 333445555 | Alice | F | 05-ABR-76 | FILHA |
| Jennifer | Wallace | 987654321 | 333445555 | Theodore | M | 25-OUT-73 | FILHO |
| Jennifer | Wallace | 987654321 | 333445555 | Joy | F | 03-MAI-48 | ESPOSA |
| Jennifer | Wallace | 987654321 | 987654321 | Abner | M | 29-FEV-78 | MARIDO |
| Jennifer | Wallace | 987654321 | 123456789 | Michael | M | 01-JAN-78 | FILHO |
| Jennifer | Wallace | 987654321 | 123456789 | Alice | F | 31-DEZ-78 | FILHA |
| Jennifer | Wallace | 987654321 | 123456789 | Elizabeth | F | 05-MAI-57 | ESPOSA |
| Joyce | English | 453453453 | 333445555 | Alice | F | 05-ABR-76 | FILHA |
| Joyce | English | 453453453 | 333445555 | Theodore | M | 25-OUT-73 | FILHO |
| Joyce | English | 453453453 | 333445555 | Joy | F | 03-MAI-48 | ESPOSA |
| Joyce | English | 453453453 | 987654321 | Abner | M | 29-FEV-78 | MARIDO |
| Joyce | English | 453453453 | 123456789 | Michael | M | 01-JAN-78 | FILHO |
| Joyce | English | 453453453 | 123456789 | Alice | F | 31-DEZ-78 | FILHA |
| Joyce | English | 453453453 | 123456789 | Elizabeth | F | 05-MAI-57 | ESPOSA |

**DEP_ATUAL**

| PNOME | SNOME | NSS | NSSEMP | NOMEDEPENDENTE | SEXO | DATANIV | RELAÇÃO |
|---|---|---|---|---|---|---|---|
| Jennifer | Wallace | 987654321 | 987654321 | Abner | M | 29-FEV-78 | MARIDO |

**RESULT**

| PNOME | SNOME | NOMEDEPENDENTE |
|---|---|---|
| Jennifer | Wallace | Abner |

**Figura 7.4 – Resultado da aplicação de uma operação de produto cartesiano**

### 7.1.5 A Operação JOIN

A operação JOIN, denotada por ⊠, é usada para combinar tuplas relacionadas de relações em uma única tupla. Esta operação é muito importante para quaisquer bases de dados relacionais, pois permite processar relacionamentos entre relações. Para ilustrar a operação JOIN, suponha que se deseja *recuperar os nomes dos gerentes de cada departamento*. Para obter-se o nome dos gerentes, é necessário combinar cada tupla de departamento com tuplas de empregados cujo valor NSS seja igual ao valor de NSSGER na tupla departamento. Isto é feito usando a operação JOIN, então projeta-se o resultado sobre aqueles atributos necessários:

$DEPT_GER \leftarrow DEPARTAMENTO \bowtie_{NSSGER=NSS} EMPREGADO$

$RESULT \leftarrow \pi_{DNOME,\ SNOME,\ PNOME} (DEPT_GER)$

O resultado da primeira operação é mostrado na Figura 7.5. O exemplo utilizado para ilustrar o CARTESIAN PRODUCT pode ser especificado usando o operador JOIN trocando as duas operações:

$EMP_DEP \leftarrow EMP_NOMES \ \chi \ DEPENDENTE$

$DEP_ATUAL \leftarrow \sigma_{NSS} =_{NSSEMP} (EMP_DEP)$

por

$DEP_ATUAL \leftarrow EMP_NOME \bowtie_{NSS=NSSEMP} DEPENDENTE$

**DEPT_GER**

| DNOME | DNÚMERO | NSSGER | ... | PNOME | MNOME | SNOME | NSS | ... |
|---|---|---|---|---|---|---|---|---|
| Pesquisa | 5 | 333445555 | ... | Franklin | T | Wong | 333445555 | ... |
| Administrativo | 4 | 987654321 | ... | Jennifer | S | Wallace | 987654321 | ... |
| Gerencial | 1 | 888665555 | ... | James | E | Borg | 888665555 | ... |

**Figura 7.5 – Resultado da execução de um operador de junção**

A forma geral da operação JOIN sobre duas relações $R(A_1, A_2, ..., A_n)$ e $S(B_1, B_2, ..., B_m)$ é:

$R \bowtie_{\text{<condição join>}} S.$

O resultado de JOIN é uma relação Q com n+m atributos $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ nesta ordem; Q tem um tupla para cada combinação de tuplas — uma de R e uma de S — *onde quer que a combinação satisfaça a condição join*. Esta é a principal diferença entre CARTESIAN PRODUCT e JOIN; em JOIN, apenas combinações de tuplas que satisfazem a condição join é que aparecerá no resultado, já no CARTESIAN PRODUCT, todas as combinações de tuplas são incluídas no resultado. A condição join é especificada sobre atributos de R e de S, e é avaliada para cada combinação de tuplas.

Uma condição join tem a forma:

<condição> AND <condição> AND,...,OR, ..., AND <condição>

onde cada condição é da forma $A_i \ \theta \ B_j$, $A_i$ é um atributo de R, $B_j$ é um atributo de S, $A_i$ e $B_j$ têm o mesmo domínio e $\theta$ é um dos operadores de comparação $\{ =, <, \leq, >, \geq, \neq \}$. Uma operação JOIN de condição especificada é denominada **THETA JOIN**. Tuplas cujos valores dos atributos join são *null* não aparecem no resultado.

É muito comum encontrar JOIN que envolva condições joins com apenas a comparação de igualdade. Um JOIN, onde apenas o operador de comparação = é utilizado é chamado **EQUIJOIN**. Os dois exemplos anteriores eram EQUIJOIN. Note-se, que no resultado de uma EQUIJOIN sempre terá um ou mais pares de atributos que tem *valores idênticos* em todas as tuplas. Por exemplo, na Figura 7.5 os valores dos atributos NSSGER e NSS são idênticos em todas as tuplas de DEPT_GER porque condição join de igualdade é especificada sobre estes

atributos. Devido a esta duplicação ser desnecessária, uma nova operação chamada **NATURAL JOIN** foi criada. Denota-se o join natural por *, que é basicamente um equijoin seguido da remoção de atributos desnecessários. Um exemplo é:

$$\text{PROJ_DEPT} \leftarrow \text{PROJETO} *_{\text{DNUM=DNÚMERO}} \text{DEPARTAMENTO}$$

Os atributos DNUM e DNÜMERO são chamados atributos de união. A relação resultante é mostrada na Figura 7.6a. Na relação PROJ_DEPT, cada tupla combina uma tupla de PROJETO com a tupla de DEPARTAMENTO que controla o projeto. Na relação resultante, manteve-se apenas o *primeiro atributo de união*. Devido às comparações em uma condição join de um join natural serem sempre comparações de igualdade, pode-se descartar o operador de comparação e apenas listar os atributos de união como segue:

$$\text{PROJ_DEPT} \leftarrow \text{PROJETO} *_{\text{(DNUM), (DNÚMERO)}} \text{DEPARTAMENTO}$$

Assim, a forma geral de um NATURAL JOIN é:

$$Q \leftarrow R *_{(\text{<lista1>}),(\text{<lista2>})} S$$

Neste caso, <lista1> especifica uma lista de i atributos de R e <lista2> especifica uma lista de i atributos de S. Apenas os atributos da primeira relação R — <lista1> — serão mantidos na relação resultante Q.

Se os atributos sobre os quais o join natural é especificado tiverem os mesmos nomes em ambas relações, pode-se tirar a condição join totalmente. Por exemplo, para aplicar um join natural sobre DNÚMERO de DEPARTAMENTO e LOCAIS_DEPTO, é suficiente escrever:

$$\text{DEPT_LOCS} \leftarrow \text{DEPARTAMENTO} * \text{LOCAIS_DEPTO}$$

A relação resultante é mostrada na Figura 7.6b, que combina cada departamento com suas localizações e tem uma tupla para cada localização. Esta operação é executada igualando-se todos os pares de atributos que tenham o mesmo nome nas duas relações.

Note que, se nenhuma combinação de tuplas satisfizer a condição join, então o resultado será uma relação vazia. Em geral, se R tiver nR tuplas e S tiver nS tuplas, o resultado de uma operação JOIN $R \bowtie_{\text{<condição join>}} S$ terá entre zero e nR*nS tuplas. Se não existir <condição join> para satisfazer, então todas as combinações de tuplas serão incluídas. Nestes casos, JOIN torna-se um CARTESIAN PRODUCT.

**PROJ_DEPT**
**(a)**

| PNOME | PNÚMERO | PLOCALIZAÇÃO | DNUM | DNOME | NSSGER | DATINICGER |
|---|---|---|---|---|---|---|
| ProdutoX | 1 | Bellaire | 5 | Pesquisa | 333445555 | 22-MAI-78 |
| ProdutoY | 2 | Sugarland | 5 | Pesquisa | 333445555 | 22-MAI-78 |
| ProdutoZ | 3 | Houston | 5 | Pesquisa | 333445555 | 22-MAI-78 |
| Automação | 10 | Stafford | 4 | Administrativo | 987654321 | 01-JAN-85 |
| Reorganização | 20 | Houston | 1 | Gerencial | 888665555 | 19-JUN-71 |
| Beneficiamento | 30 | Stafford | 4 | Administrativo | 987654321 | 01-JAN-85 |

**DEPT_LOCS**
**(b)**

| DNOME | DNÚMERO | NSSGER | DATINICGER | DLOCALIZAÇÃO |
|---|---|---|---|---|
| Gerencial | 1 | 888665555 | 19-JUN-71 | Houston |
| Administrativo | 4 | 987654321 | 01-JAN-85 | Stafford |
| Pesquisa | 5 | 333445555 | 22-MAI-78 | Bellaire |
| Pesquisa | 5 | 333445555 | 22-MAI-78 | Sugariand |
| Pesquisa | 5 | 333445555 | 22-MAI-78 | Houston |

**Figura 7.6 – Resultado da execução de um operador Natural Join.**

### 7.1.6 Conjunto completo de Operações da Álgebra Relacional

Será mostrado que o conjunto de operações da álgebra relacional {$\sigma$, $\pi$, $\cup$, -, $\chi$} é um conjunto completo; isto é, quaisquer outras operações da álgebra relacional podem ser expressas como uma seqüência de operações deste conjunto. Por exemplo, a operação de INTERSECTION pode ser expressa usando UNION e DIFFERENCE como segue:

$R \cap S \equiv (R \cup S)-((R - S) \cup (S - R))$

Embora, estritamente falando, a INTERSECTION, seja desnecessária, é inconveniente especificar esta complexa expressão todas as vezes que for necessário utilizar a interseção. Como um outro exemplo, uma operação JOIN pode ser especificada como uma CARTESIAN PRODUCT seguida pela operação SELECT como discutido anteriormente:

$R \boxtimes_{\text{<condição>}} S \equiv \sigma_{\text{<condição>}} (R \ \chi \ S)$

Similarmente, uma operação NATURAL JOIN pode ser especificada como um CARTESIAN PRODUCT seguida pelas operações SELECT e PROJECT. Assim, as várias formas de JOIN também não são estritamente necessárias para expressar o poder da álgebra relacional. No entanto, elas são muito convenientes — assim como a operação INTERSECTION — porque elas são utilizadas com freqüência pelas aplicações de base de dados relacional. Outras operações foram incluídas por conveniência. Será discutida uma delas: DIVISION.

### 7.1.7 A Operação DIVISION

A operação DIVISION é útil para um tipo especial de consulta que ocorre freqüentemente em aplicações de base de dados. Esta requisição pode ser ilustrada pela seguinte consulta: "Recupere os nomes dos empregados que trabalham em todos os projetos em que 'John Smith' trabalha”. Para expressar esta consulta usando DIVISION deve-se fazer o seguinte: primeiro recuperar a lista de números de projetos em que 'John Smith' trabalha em uma relação intermediária SMITH_PNOS:

$SMITH \leftarrow \sigma_{\text{PNOME = 'John' AND SNOME = 'Smith'}} (EMPREGADO)$

$SMITH_PNOS \leftarrow \pi_{\text{PNRO}} (TRABALHA_EM *_{\text{NSSEMP = NSS}} SMITH)$

Depois, criar uma relação que inclua tuplas da forma <PNRO, NSSEMP> que lista todos os empregados, cujo número do seguro social é NSSEMP, que trabalham num determinado projeto PNRO:

$NSS_PNRO \leftarrow \pi_{\text{PNRO, NSSEMP}} (TRABALHA_EM)$

Finalmente, aplicar a operação DIVISION para as relações obtidas a fim de obter os números dos seguros sociais desejados:

$NSS_DESEJADO(NSS) \leftarrow NSS_PNRO \div SMITH_PNOS$

$RESULT \leftarrow \pi_{\text{PNOME, SNOME}} (NSS_DESEJADO * EMPREGADO)$

Os resultados das operações acima são mostrados na Figura 7.7a. Em geral, a operação DIVISION é aplicada para duas relações $R(Z) \div S(X)$, onde $X \subseteq Z$. Seja $Y=Z - X$; isto é, Y é o conjunto de atributos de R que não são atributos de S. O resultado da divisão é uma relação T(Y) que incluirá uma tupla t se tR cujo tR[Y] = t aparecer em R com tR[X] = Ts *para toda tupla* tS em S. Isto significa que para uma tupla aparecer no resultado T da divisão, os valores em t devem aparecer em R em combinação com *todas as tuplas* de S.

A Figura 7.7b ilustra a utilização da operação DIVISON onde R = Z = {A, B}, S = X = {A}, e T = Y = {B}; isto é, X e Y são (ambos) atributos simples. A relação resultante T terá apenas um único atributo B = Z - X. Note que b1 e b4 aparecem em R em combinação com todas as três tuplas de S; por isso é que eles aparecem na relação resultado T. Todos os outros valores de B em R não apareceram com todas as tuplas de S e não foram selecionados; b2 não aparece com a2; e b3 não aparece com a1.

A operação de divisão pode ser expressa como uma seqüência de operações $\pi$, $\chi$ e -:

$$T1 \leftarrow \pi_Y (R)$$

$$T2 \leftarrow \pi_Y ((S \chi T1) - R)$$

$$T \leftarrow T1 - T2$$

(a)

| NSS_PNRO | NSSEMP | PNRO |
|---|---|---|
| | 123456789 | 1 |
| | 123456789 | 2 |
| | 666884444 | 3 |
| | 453453453 | 1 |
| | 453453453 | 2 |
| | 333445555 | 2 |
| | 333445555 | 3 |
| | 333445555 | 10 |
| | 333445555 | 20 |
| | 999887777 | 30 |
| | 999887777 | 10 |
| | 987987987 | 10 |
| | 987987987 | 30 |
| | 987654321 | 30 |
| | 987654321 | 20 |
| | 8886655555 | 20 |

| SMITH_PNO | PNRO |
|---|---|
| | 1 |
| | 2 |

| NSS_DESEJADO | NSS |
|---|---|
| | 123456789 |
| | 453453453 |

(b)

| R | A | B |
|---|---|---|
| | a1 | b1 |
| | a2 | b1 |
| | a3 | b1 |
| | a4 | b1 |
| | a1 | b2 |
| | a3 | b2 |
| | a2 | b3 |
| | a3 | b3 |
| | a4 | b3 |
| | a1 | b4 |
| | a2 | b4 |
| | a3 | b4 |

| S | A |
|---|---|
| | a1 |
| | a2 |
| | a3 |

| T | B |
|---|---|
| | b1 |
| | b4 |

**Figura 7.7 – Resultado da aplicação de um operador de divisão**

### 7.1.8 Operações Relacionais Adicionais

Existem algumas consultas comuns em bases de dados relacionais que não podem ser realizadas como as operações da álgebra descritas na seção anterior. Muitas linguagens de consulta de SGBD possuem a capacidade de realizar essas consultas. Nesta seção serão discutidas algumas dessas consultas e define-se operações adicionais que são usadas para expressar essas consultas.

### 7.1.9 Funções de Agregação

O primeiro tipo de consulta que não pode ser expressa na álgebra relacional é conhecido como **funções agregadas** sobre coleções de valores da base de dados. Por exemplo, pode-se querer recuperar a média ou total salarial de todos os empregados ou o número de tuplas de empregados. As funções normalmente aplicadas para coleções de valores numéricos são:

SUM, AVERAGE, MAXIMUM e MINIMUM. A função de contagem de tuplas é normalmente chamada COUNT. Cada uma destas funções pode ser aplicada a todas as tuplas de uma relação.

Um outro tipo comum de consulta envolve o agrupamento de tuplas de uma relação pelo valor de alguns de seus atributos e então a aplicação de alguma função agregada independente para cada grupo de tuplas. Um exemplo é o agrupamento de tuplas de empregados pelo NDEP. Pode-se então listar, para cada NDEP a média salarial de empregados num determinado departamento.

Pode-se definir uma operação FUNCTION, que pode ser usada para especificar estes tipos de consultas.

<atributos de agrupamento> $\Im$ <lista de funções> (<nome da relação>)

onde <atributos de agrupamento> é uma lista de atributos da relação especificada em <nome da relação> e <lista de funções> é uma lista de pares (<função><atributo>). Em cada par, <função> é uma das funções seguintes: SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT, e <atributo> é um atributo da relação especificada por <nome da relação>. A relação resultante tem tantos atributos quanto aqueles que existirem em atributos de agrupamento além da lista de funções combinada. Por exemplo, para recuperar cada número de departamento, o número de empregados em departamento, e sua média salarial, escreve-se:

R(DNO, NRO_EMPS, MÉDIA) $\leftarrow_{\text{NDEP}} \Im_{\text{COUNT NSS, AVERAGE SALÁRIO}}$ (EMPREGADO)

O resultado desta operação é mostrado na Figura 7.8a.
No exemplo acima, foi especificada uma lista de nomes de atributos - entre parênteses - para a relação R. Se nenhuma lista tivesse sido especificada, então os atributos da relação resultante teriam como o nome a concatenação do nome da função com o nome do atributo especificado. Por exemplo, o resultado da seguinte operação é mostrado na Figura 7.8b.

$_{\text{NDEP}} \Im_{\text{COUNT NSS, AVERAGE SALÁRIO}}$ (EMPREGADO)

Se nenhum atributo de agrupamento for especificado, as funções serão aplicadas *para todos os valores* de tuplas na relação, tal que a relação resultante *terá uma única tupla*. Por exemplo, o resultado da seguinte operação é mostrado na Figura 7.8

$\Im_{\text{COUNT NSS, AVERAGE SALÁRIO}}$ (EMPREGADO)

É importante destacar que o resultado da aplicação de qualquer função de agregação sempre será uma relação e não um número.

(a)

| R | DNO | NRO_EMPS | MÉDIA |
|---|---|---|---|
| | 5 | 4 | 3325 |
| | 4 | 3 | 3100 |
| | 1 | 1 | 5500 |

(b)

| NDEP | COUNT_NSS | AVERAGE_SALÁRIO |
|---|---|---|
| 5 | 4 | 3325 |
| 4 | 3 | 3100 |
| 1 | 1 | 5500 |

(c)

| COUNT_NSS | AVERAGE_SALÁRIO |
|---|---|
| 8 | 3512.5 |

**Figura 7.8 – Exemplo de aplicação de funções de agregação**

### 7.1.10 Operações de Fechamento Recursivo (Clausura Recursiva)

Um outro tipo de operação que, em geral, não pode ser definido na álgebra relacional é a **fechamento recursivo**. Esta operação é aplicada em relacionamentos recursivos, como verificado no relacionamento entre empregado e supervisor.

Este relacionamento é descrito por uma chave-estrangeira NSSSUPER da relação empregado nas Figura 5.4 e 5.5, que relaciona cada tupla de empregado (no papel de supervisionado) a uma outra tupla empregado (no papel de supervisor). Um exemplo de uma operação recursiva é recuperar todos os supervisionados de um empregado **e** em todos os níveis, isto é, todos os empregados **e'** diretamente supervisionados por **e**, todos os empregados **e''** diretamente supervisionados por **e'**, todos os empregados **e'''** diretamente supervisionados por **e''**, e assim por diante. Embora seja simples especificar, na álgebra relacional, todos os empregados supervisionados por **e** num nível específico, é difícil especificar todos os supervisionados em todos os níveis. Por exemplo, para especificar os NSSs de todos os empregados **e'** diretamente supervisionados — no nível um — por **e** cujo nome é 'James Borg' (veja Figura 5.4), pode-se aplicar as seguintes operações:

$\text{BORG_NSS} \leftarrow \pi_{\text{NSS}} (\sigma_{\text{PNOME = 'James' AND SNOME = 'Borg'}} (\text{EMPREGADO}))$

$\text{SUPERVISÃO(SNN1, SNN2)} \leftarrow \pi_{\text{NSS, NSSSUPER}} (\text{EMPREGADO})$

$\text{RESULT1} \leftarrow \pi_{\text{SNN1}} (\text{SUPERVISÃO} \boxtimes_{\text{SNN2 = NSS}} \text{BORG_NSS})$

Para recuperar todos os empregados supervisionados por Borg no nível dois, isto é, todos os empregados **e''** supervisionados pelo empregado **e'** que é diretamente supervisionado por Borg, pode-se aplicar um outro JOIN ao resultado da primeira consulta:

$\text{RESULT2} \leftarrow \pi_{\text{SNN1}} (\text{SUPERVISÃO} \boxtimes_{\text{SNN2 = NSS}} \text{RESULT1})$

Para obter todos os empregados supervisionados nos níveis um e dois por 'James Borg', pode-se aplicar a operação UNION como segue:

$\text{RESULT3} \leftarrow (\text{RESULT1} \cup \text{RESULT2})$

Embora seja possível recuperar os empregados supervisionados em cada nível, não é possível especificar uma consulta tal como " recupere todos os supervisionados de 'James Borg' em todos os níveis" se não for conhecido o número máximo de níveis.

## 7.2 Exemplos de Consultas na Álgebra Relacional

Nesta seção serão apresentados exemplos para ilustrar o uso das operações da álgebra relacional. Todos os exemplos referem-se à base de dados da Figura 5.4. Em geral, a mesma consulta pode ser realizada de várias maneiras usando diferentes operadores relacionais.

**Consulta 1**: Encontrar o nome e o endereço de todos os empregados que trabalham para o departamento 'Pesquisa'.

$\text{PESQUISA_DEPTO} \leftarrow \sigma_{\text{DNOME = 'Pesquisa'}} (\text{DEPARTAMENTO})$

$\text{PESQUISA_DEPTO_EMPS} \leftarrow (\text{PESQUISA_DEPTO} \boxtimes_{\text{DNÚMERO = NDEP}} \text{EMPREGADO})$

$\text{RESULT} \leftarrow \pi_{\text{PNOME, SNOME, ENDEREÇO}} (\text{PESQUISA_DEPTO_EMPS})$

Esta consulta poderia ser especificada de outra maneira; por exemplo, a ordem das operações JOIN e SELECT poderia ser invertida ou o JOIN poderia ser trocado pelo NATURAL JOIN.

**Consulta 2**: Para todo projeto localizado em 'Stafford', listar o número do projeto, o número do departamento responsável, e o sobrenome, endereço e data de nascimento do gerente responsável pelo departamento.

STAFFORD_PROJS $\leftarrow \sigma_{\text{PLOCALIZAÇÃO = 'Stafford'}}$ (PROJETO)
CONTR_DEPT $\leftarrow$ (STAFFORD_PROJS $\bowtie_{\text{DNUM = DNÚMERO}}$ DEPARTAMENTO)
PROJ_DEPT_MGR $\leftarrow$ (CONTR_DEPT $\bowtie_{\text{SSNGER = NSS}}$ EMPREGADO)
RESULT $\leftarrow \pi_{\text{PNÚMERO, DNUM, SNOME, ENDEREÇO, DATANASC}}$ (PROJ_DEPT_MGR)

**Consulta 3**: Encontrar os nomes de empregados que trabalham em todos os projetos controlados pelo departamento 5.

DEPT5_PROJS(PNO) $\leftarrow \pi_{\text{PNÚMERO}}$ ($\sigma_{\text{DNUM=5}}$ (PROJETO)))
EMP_PROJ(NSS, PNO) $\leftarrow \pi_{\text{NSSEMP, PNRO}}$ (TRABALHA_EM)
RESULT_EMP_SSNS $\leftarrow$ EMP_PROJ $\div$ DEPT5_PROJS
RESULT $\leftarrow \pi_{\text{SNOME, PNOME}}$ (RESULT_EMP_SSNS * EMPREGADO)

**Consulta 4**: Fazer uma lista de números de projetos no qual um empregado, cujo sobrenome é 'Smith' , trabalha no projeto ou é gerente do departamento que controla o projeto.

SMITH(NSSEMP) $\leftarrow \pi_{\text{NSS}}$ ($\sigma_{\text{SNOME='Smith'}}$ (EMPREGADO))
SMITH_WORKER_PROJS $\leftarrow \pi_{\text{PNRO}}$ (TRABALHA_EM * SMITH)
MGRS $\leftarrow \pi_{\text{SNOME, DNÚMERO}}$ (EMPREGADO $\bowtie_{\text{NSS = NSSGER}}$ DEPARTAMENTO)
SMITH_MGS $\leftarrow \sigma_{\text{SNOME = 'Smith'}}$ (MGRS)
SMITH_MANAGED_DEPTS(DNUM) $\leftarrow \pi_{\text{DNÚMERO}}$ (SMITH_MGRS)
SMITH_MGR_PROJS(PNRO) $\leftarrow \pi_{\text{PNÚMERO}}$ (SMITH_MANAGED_DEPTS * PROJETO)
RESULT $\leftarrow$ (SMITH_WORKER_PROJS $\cup$ SMITH_MGR_PROJS)

**Consulta 5**: Listar os nomes de todos os empregados com dois ou mais dependentes.

Esta consulta não pode ser realizada usando apenas a álgebra relacional. Deve-se utilizar a operação FUNCTION com a função de agregação COUNT, que não é da álgebra relacional. Nas formulações que se seguirão, assume-se que dependentes de um mesmo empregado possuem nomes distintos.

T1(NSS, NO_DE_DEPS) $\leftarrow_{\text{NSSEMP}} \Im_{\text{COUNT NOMEDEPENDENTE}}$ (DEPENDENTE)
T2 $\leftarrow \sigma_{\text{NO_DE_DEPS >= 2}}$ (T1)
RESULT $\leftarrow \pi_{\text{SNOME, PNOME}}$ (T2 * EMPREGADO)

**Consulta 6**: Listar os nomes dos empregados que não possuem dependentes.

TODOS_EMPS $\leftarrow \pi_{\text{NSS}}$ (EMPREGADO)
EMPS_COM_DEPS(NSS) $\leftarrow \pi_{\text{NSSEMP}}$ (DEPENDENTE)
EMPS_SEM_DEPS $\leftarrow$ (TODOS_EMPS - EMPS_COM_DEPS)
RESULT $\leftarrow \pi_{\text{SNOME, PNOME}}$ (EMPS_SEM_DEPS * EMPREGADO)

**Consulta 7**: Listar os nomes dos gerentes que têm ao menos um dependente.

MGRS(NSS)$\leftarrow\pi_{NSSGER}$ (DEPARTAMENTO)
EMPS_COM_DEPS(NSS)$\leftarrow\pi_{NSSEMP}$ (DEPENDENTE)
MGRS_COM_DEPS$\leftarrow$(MGRS $\cap$ EMPS_COM_DEPS)
RESULT$\leftarrow\pi_{SNOME, PNOME}$ (MGRS_COM_DEPS * EMPREGADO)

## 7.3 Questões de Revisão

1. O que é união compatível? Por que as operações UNION, INTERSECTION e DIFFERENCE são operações que necessitam que as relações sejam: união compatível?
2. Discuta algumas das consultas onde seja necessário renomear atributos a fim de especificar consultas não ambíguas.
3. Discuta os vários tipos de operações JOIN. Resuma em forma de tabela que contenha o nome da operação, o propósito da operação e a notação.
4. Especifique as seguintes consultas sobre a base de dados mostrada na Figura 5.3 usando operações relacionais discutidas neste capítulo. Mostra também os resultados de cada consulta se aplicada à base de dados da Figura 5.4.

    a) Recuperar os nomes de empregados do departamento 5 que trabalham mais que 10 horas no projeto 'ProdutoX'.
    b) Listar os nomes dos empregados que tenham um dependente com o mesmo nome (PNOME).
    c) Encontrar os nomes de empregados que são diretamente supervisionados por 'Franklin Wong'.
    d) Para cada projeto, listar o nome do projeto e o total de horas (de todos os empregados) gastos em cada projeto.
    e) Recuperar os nomes dos empregados que trabalham em todos os projetos.
    f) Recuperar os nomes dos empregados que não trabalham em quaisquer projetos.
    g) Para cada departamento, recuperar o nome do departamento e a média salarial dos empregados que trabalham no departamento.
    h) Recuperar a média salarial de todos os empregados femininos.
    i) Encontrar os nomes e endereços de empregados que trabalham em ao menos um projeto localizado em Houston mas cujo departamento não possua localização em Houston.
    j) Listar os sobrenomes dos gerentes de departamentos que não tenham dependentes.
    k) Generalize a consulta i) acima para listar os nomes e endereços de empregados que trabalham em um projeto em alguma cidade , mas que o departamento não tenha nenhuma localização nessa cidade.

## *7.4 Cálculo Relacional*

O Cálculo Relacional (CR) é uma linguagem de consulta formal onde, por meio de uma expressão declarativa, pode-se especificar uma solicitação de recuperação. Não há nenhuma restrição na forma de avaliar uma solicitação.
Uma expressão de cálculo permite a descrição da consulta desejada sem especificar os procedimentos para obtenção dessas informações, ou seja, é não-procedural. Contudo, deve ser capaz de descrever formalmente a informação desejada, com exatidão.
Existem dois tipos: Cálculo Relacional de Tuplas (CRT) e Cálculo Relacional de Domínio (CRD), ambos subconjuntos simples de lógica de primeira ordem.
No Cálculo Relacional existem variáveis, constantes, operadores lógicos, de comparação e quantificadores. As expressões de Cálculo são chamadas de fórmulas. Uma tupla de respostas é essencialmente uma atribuição de constantes às variáveis que levam a fórmula a um estado verdadeiro.
Em CRT, as variáveis são definidas sobre (isto é, associam) tuplas. Já em CRD, variáveis são definidas sobre o domínio dos elementos (ou seja, sobre os valores dos campos).
Todas as expressões de consulta descritas em CR possuem equivalentes em AR.

### 7.4.1 Cálculo Relacional de Tuplas

É baseado na especificação de um número de variáveis de tuplas. Cada variável de tupla pode assumir como seu valor qualquer tupla da relação especificada.

Uma consulta em CRT é especificada da seguinte forma:

{variável tupla | predicado}

O resultado de tal consulta é o conjunto de todas as variáveis tuplas para as quais o predicado é indicado como verdadeiro.

Uma expressão genérica do cálculo relacional de tuplas tem a forma$\{t_1.A_1, t_2.A_2,..., t_n.A_n \mid predicado(t_1, t_2,..., t_n, t_{n+1}, t_{n+2}, ...,t_{n+m})\}$

onde $t_1, t_2,..., t_n, t_{n+1}, t_{n+2}, ...,t_{n+m}$ são variáveis de tuplas, cada $A_i$ é um atributo da relação na qual $t_i$ se encontra e o predicado é uma fórmula do cálculo relacional de tuplas.

As fórmulas atômicas de cálculo de predicados podem ser uma das seguintes:
1-) Uma fórmula atômica R(ti), onde R é o nome de uma relação e ti é uma variável de tupla. Este átomo identifica a extensão da variável de tupla ti como a relação cujo nome seja R.
2-) Uma fórmula atômica ti.A op tj.B, onde op é um dos operadores de comparação no conjunto {=, >, <, ...}, ti e tj são variáveis de tuplas, A é um atributo da relação na qual ti se encontra, B é um atributo da relação na qual tj se encontra.
3-) Um fórmula atômica ti.A op c ou c op tj.B, onde op é um dos operadores de comparação no conjunto {=, >, <, ...}, ti e tj são variáveis de tuplas, A é um atributo da relação na qual ti se encontra, B é um atributo da relação na qual tj se encontra e c é um valor constante.

Cada uma das fórmulas atômicas anteriormente especificadas tem seu valor verdade avaliado como TRUE ou FALSE para uma combinação específica de tuplas. Para fórmulas do tipo 1, caso a variável de tupla seja atribuída a uma tupla da relação R dada, esta assume valor TRUE; caso contrário, FALSE. Já nas fórmulas do tipo 2 e 3, se as variáveis de tupla forem designadas de forma que os valores dos atributos especificados satisfaçam o predicado, esta assumirá valor verdade TRUE.

Todas as variáveis tuplas abordadas até então são consideradas variáveis livres[4], uma vez que estas não aparecem quantificadas. Contudo, além das definições acima, quantificadores (universal (∀) ou existencial (∃)) podem aparecer nas fórmulas. Neste caso, as variáveis que os sucedem são denominadas variáveis limite.

Uma fórmula é definida, de forma recursiva, por uma ou mais fórmulas atômicas, conectadas por operadores lógicos (AND, OR, NOT), como segue:

1-) Qualquer fórmula atômica.

2-) Se F1 e F2 são fórmulas atômicas, então (F1 AND F2), (F1 OR F2), NOT (F1) e NOT (F2) também o são, tendo seus valores verdade derivados a partir de F1 e F2, da seguinte forma:

a-) (F1 AND F2) será TRUE apenas se ambos, F1 e F2, forem TRUE;
b-) (F1 OR F2) será TRUE quando uma das duas fórmulas F1 e F2, for TRUE;
c-) NOT(F1) será TRUE quando F1 for FALSE;
c-) NOT(F2) será TRUE quando F2 for FALSE.

3-) Se F1 é uma fórmula atômica, então (∃ t[5])(F1) também o é, e seu valor verdade apenas será TRUE se a fórmula F for avaliada como verdadeira para pelo menos uma tupla atribuída para ocorrências livres de t em F.

4-) Se F1 é uma fórmula atômica, então (∀ t)(F1) também o é, e seu valor verdade apenas será TRUE se a fórmula F for avaliada como verdadeira para todas as tuplas atribuídas para ocorrências livres de t em F.

É possível escrever expressões equivalentes manipulando operadores e quantificadores. Alguns casos dessa manipulação podem ser declarados da seguinte forma:

1-) F1 AND F2 ≡ NOT(NOT F1 OR NOT F2).
2-) (∀ t) ∈ r (F1(t)) ≡ NOT (∃ t) ∈ r (NOT F1(t)).
3-) F1 ⇒ F2 ≡ NOT F1 OR F2.
4-) (∀ x) (F(x)) ≡ NOT (∃ x) (NOT (F(x))
5-) (∃ x) (F(x)) ≡ NOT (∀ x) (NOT (F(x))
6-) (∀ x) (F(x) AND P(x)) ≡ NOT (∃ x) (NOT (F(x)) OR NOT (P(x)))
7-) (∀ x) (F(x) OR P(x)) ≡ NOT (∃ x) (NOT (F(x)) AND NOT (P(x)))
8-) (∃ x) (F(x) OR P(x)) ≡ NOT (∀ x) (NOT (F(x)) AND NOT (P(x)))
9-) (∃ x) (F(x) AND P(x)) ≡ NOT (∀ x) (NOT (F(x)) OR NOT (P(x)))

Abaixo seguem exemplos de consultas em CRT.

1-) Encontre todos os empregados cujos salários estejam acima de R$3.500,00.

{t | EMPREGADO(t) AND t.SALARIO > 3500}

2-) Dê apenas os nomes e sobrenomes dos empregados cujos salários estejam acima de R$3.500,00.

{t.NOME, t.SOBRENOME | EMPREGADO(t) AND t.SALARIO > 3500}

3-) Selecione o nome e o endereço dos empregados que trabalham para o departamento de 'Informática'.

{t.NOME, t.SOBRENOME, t.ENDERECO | EMPREGADO(t) AND (∃ d) (DEPARTAMENTO (d) AND d.NOMED = 'Informática' AND d.NUMERODEP = t.NUD)}

---

[4] As únicas variáveis de tupla livres em uma expressão de cálculo relacional devem ser aquelas à esquerda da barra ( | ).
[5] t é uma variável de tupla.

4-) Para cada projeto localizado em 'São Paulo', liste o número do mesmo, o nome do departamento proponente, bem como sobrenome, data de nascimento e endereço do gerente responsável.

{p.NUMEROP, p.NUMD, m.SOBRENOME, m.DATANASCIMENTO, m.ENDERECO | PROJETO(p) AND EMPREGADO(m) AND p.LOCALIZACAO = 'São Paulo' AND (($\exists$ d) (DEPARTAMENTO(d) AND d.NUMD = d.NUMERODEP AND d.NSSGER = m.NSS) )}

5-) Encontre os nomes dos empregados que trabalham em todos os projetos controlados pelo departamento de número 5.

{e.SOBRENOME, e.NOME | EMPREGADO(e) AND (($\forall$ x) (NOT(PROJETO(x)) OR NOT(x.NUMD = 5) OR (($\exists$ w) (TRABALHA_EM(w) AND w.NSSE = e.NUMEROP AND x.NUMEROP = w.NUMP))))}

**Expressões Seguras**

Uma expressão em CRT pode gerar uma infinidade de relações. Para a expressão

{t | NOT (R(t))}

pode existir uma infinidade de tuplas que não estão em R, de forma que esta é não-segura. A maioria dessas tuplas contém valores que não estão no banco de dados, logo, não são desejáveis como resultados.

Uma expressão segura no CR é uma expressão que garante a produção de um número finito de tuplas como resultado.

Para melhor definir expressão segura, o conceito de domínio pode ser utilizado. O domínio de uma expressão P é o conjunto de todos os valores referenciados por P. Isso inclui os valores mencionados em P propriamente dito, assim como os valores que aparecem na tupla de uma relação referenciada por P. Assim, o domínio de P é um conjunto de valores que aparecem explicitamente em P ou que aparecem em uma ou mais relações cujos nomes aparecem em P.

## 7.4.2 Cálculo Relacional de Domínio

A diferença básica entre CRT e CRD é que neste último as variáveis estendem-se sobre valores únicos de domínios de atributos. Para formar uma relação de grau n para um resultado de consulta, faz-se necessário criar n variáveis de domínio, uma para cada atributo.

Uma expressão genérica do cálculo relacional de tuplas tem a forma

$\{x_1, x_2,..., x_n \mid predicado(x_1, x_2,..., x_n, x_{n+1}, x_{n+2}, ...,x_{n+m})\}$

onde $x_1, x_2,..., x_n, x_{n+1}, x_{n+2}, ...,x_{n+m}$ são variáveis de domínio aplicadas sobre o domínio dos atributos requeridos na consulta e predicado é uma fórmula atômica do CRD, que pode ser especificada em uma das formas que segue:

1-) Uma fórmula atômica $R(x_1, x_2,..., x_n)$, onde R é o nome de uma relação de grau j e cada $x_i$, $1 \leq i \leq j$, é uma variável de domínio. Isto implica que uma lista de valores de $<x_1, x_2,..., x_j>$ deve ser uma tupla na relação R, onde $x_i$ é o valor do i-ésimo valor de atributo da tupla.

2-) Uma fórmula atômica xi op xj, onde op é um operador de comparação {=, <, >, ...} e xi e xj são variáveis de domínio.

3-) Uma fórmula atômica xi op c ou c xj, onde op é um operador de comparação {=, <, >, ...} e xi e xj são variáveis de domínio e c é um valor constante qualquer.

Como em CRT, as fórmulas são avaliadas em valores verdade para um conjunto específico de valores. Para a fórmula do tipo 1, o valor verdade será TRUE apenas se houver valores de domínio correspondentes a uma tupla de R atribuídos às variáveis de domínio que representam. Para os casos 2 e 3, o valor verdade será TRUE caso as variáveis de domínio possuam valores que satisfaçam o predicado.

**Expressões Seguras**

Uma expressão em CRD é dita segura se:

1-) Todos os valores que aparecem nas tuplas da expressão são valores dentro do domínio da mesma.
2-) Para todas as sub-fórmulas (∃ x) (P(x)), a sub-fórmula é verdadeira se, e somente se, existir um valor x no domínio de P tal que P(x) seja verdadeiro.
3-) Para toda sub-fórmula (∀ x) (P(x)), a sub-fórmula é verdadeira se, e somente se, P(x) for verdadeiro para todos os valores de x dentro do domínio de P.

As proposições acima garantem que possamos testar todas as sub-fórmulas “existe um” e “para todo” sem a necessidade de testar todas as suas infinitas possibilidades de ocorrência.

Abaixo, para fins de comparação, seguem em CRD os mesmos exemplos de consultas já escritos em CRT.

1-) Encontre todos os empregados cujos salários estejam acima de R$3.500,00.

{qrstuvwxyz **|** (∃ x) EMPREGADO(qrstuvwxyz) AND x > 3500}

2-) Dê apenas os nomes e sobrenomes dos empregados cujos salários estejam acima de R$3.500,00.

{qs | (∃ x) EMPREGADO(qrstuvwxyz) AND x > 3500}

3-) Selecione o nome e o endereço dos empregados que trabalham para o departamento de ‘Informática’.

{qsv | (∃ z) (∃ l) (∃ m) (EMPREGADO(qrstuvwxyz) AND DEPARTAMENTO(lmno) AND l = ‘Informática’ AND m = z)}

4-) Para cada projeto localizado em ‘São Paulo’, liste o número do mesmo, o nome do departamento proponente, bem como sobrenome, data de nascimento e endereço do gerente responsável.

{iksuv | (∃ j) (∃ m) (∃ n) (∃ t) (PROJETO(hijk) AND EMPREGADO(qrstuvwxyz) AND DEPARTAMENTO(lmno) AND k = m AND n = t AND j = ‘São Paulo’)}

5-) Encontre os nomes dos empregados que trabalham em todos os projetos controlados pelo departamento de número 5.

{qs | (∃ a) (∃ t) (∃ b) (∃ z) EMPREGADO(qrstuvwxyz) AND ((∀ i) (NOT(PROJETO(hijk)) OR NOT(k = 5) OR (TRABALHA_EM(abc) AND a = t AND i = b))))}

# 8 A Linguagem SQL

A linguagem SQL é um padrão de linguagem de consulta comercial que usa uma combinação de construtores em Álgebra e Cálculo Relacional e possui as seguintes partes:

- Linguagem de definição de dados (DDL)
- Linguagem interativa de manipulação de dados (DML)
- Incorporação DML (*Embedded* SQL)
- Definição de Visões
- Autorização
- Integridade
- Controle de Transações

O SQL permite que uma tabela (relação) tenha duas ou mais tuplas idênticas em todos os seus valores de atributos. Assim, em geral, uma relação SQL não é um conjunto de tuplas porque um conjunto não permite elementos idênticos. Ao invés disso, SQL é um multi-conjunto (algumas vezes chamado de *bag*) de tuplas.

## 8.1 Estrutura Básica

Uma expressão básica em SQL consiste em três cláusulas: select, from e where.

A cláusula **SELECT** corresponde à operação de projeção da álgebra relacional. É usada para relacionar os atributos desejados no resultado de uma consulta.

O resultado de uma consulta SQL é, obviamente, uma relação. SQL permite duplicidades nas tuplas de resposta.Quando desejamos forçar a eliminação de duplicidade, podemos inserir a palavra chave DISTINCT depois de SELECT.

Para especificar explicitamente que as duplicidades não serão eliminadas a SQL nos permite usar a palavra-chave all depois de select. Uma vez que a manutenção de duplicidade é padrão, o uso de all é facultativo.

O asterisco "*" pode ser usado para denotar "todos os atributos".

A cláusula **FROM** corresponde à operação de produto cartesiano da álgebra relacional. Ela associa as relações que serão pesquisadas durante a avaliação de uma expressão.

A cláusula **WHERE** corresponde à seleção do predicado da álgebra relacional. Consiste em um predicado envolvendo atributos da relação que aparece na cláusula FROM.

A cláusula where pode conter expressões aritméticas envolvendo os operadores de comparação <, <=, >, >=, = e <>, e operandos constantes ou atributos das tuplas.

Em SQL, nessa cláusula pode-se usar os conectores lógicos AND, OR e NOT;

SQL possui o operador de comparação BETWEEN AND para simplificar a cláusula where que especifica que um valor pode ser menor ou igual a algum valor e maior ou igual a algum outro valor.

Uma consulta típica em SQL tem a forma:

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_n$
where P

Cada $A_i$ representa um atributo e cada $r_i$, uma relação. P é um predicado a ser satisfeito. Esta consulta equivale à seguinte expressão em AR:

$\pi_{A_1, A_2, ..., A_n} (\sigma_p (r_1 \times r_2 \times ... r_m))$

A SQL forma um produto cartesiano das relações indicadas na cláusula FROM, executa uma seleção em AR usando o predicado da cláusula WHERE e, então, projeta o resultado nos atributos da cláusula SELECT. Na prática, ela pode converter em outra expressão equivalente que seja mais eficiente no processamento.

### 8.1.1 A operação RENAME

SQL proporciona um mecanismo para renomear tanto atributos quanto relações, usando a cláusula AS, da seguinte forma:

nome_antigo as nome_novo

Pode aparecer tanto na cláusula SELECT quanto na cláusula FROM.
A cláusula **AS** é particularmente útil na definição de variável tupla (quando utilizada na cláusula FROM), como é feito no cálculo relacional. Uma variável tupla em SQL precisa estar associada a uma relação.

### 8.1.2 Operações com Strings

As operações em strings mais usadas são as checagens para verificação de coincidências de pares, utilizando o operador LIKE. Esses pares são identificados por meio do uso de dois caracteres especiais:

Porcentagem ( % ): compara qualquer string;
Sublinhado ( _ ): compara qualquer caracter.

Exemplos:
"_ _ _ _%" corresponde a qualquer string com pelo menos quatro caracteres.
"Uni%" corresponde a qualquer string que comece com "Uni", como, "universo", "universal", "universidade".

Utilizando not LIKE pode-se pesquisar diferenças, ao invés de coincidências.

Obs.: Essas comparações são *case sensitive*.

### 8.1.3 Ordenação e Apresentação de Tuplas

A linguagem de consulta SQL possui a cláusula ORDER BY que possibilita a apresentação dos resultados de uma consulta segundo uma determinada ordem de tuplas.
Para completar uma solicitação de ORDER BY a SQL precisa realizar uma SORT (intercalação).
Uma intercalação com muitas tuplas pode ser custosa, portanto, só usar quando necessária.
Para listarmos uma relação em ordem ascendente usamos order by nome do campo asc. Em ordem descendente, usamos order by nome do campo desc.
Por default, a relação dos itens é efetuada em ordem ascendente.

### 8.1.4 Operações com Conjuntos

SQL possui as operações de união (union), interseção (intersect) e exceto (except), que correspondem às operações $\cup$, $\cap$ e – álgebra relacional.

Contudo, existem várias restrições para o uso destes operadores e certos produtos não oferecem suporte para estas operações.

### 8.1.5 Funções Agregadas

Funções que tomam uma coleção (um conjunto ou subconjunto) de valores como entrada, retornando um único valor.
Funções ofertadas pela SQL:
Sum: soma/total
avg: média
count: contagem
Min: mínimo
Max: Maximo

A entrada para sum e avg precisa, obrigatoriamente, ser um conjunto de números, mas as demais operações não impõem esta restrição.
Exemplo:
select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P
group by $A_1$;

Por vezes, precisamos aplicar uma função agregada não apenas a um conjunto de tuplas, mas também a um grupo de conjunto de tuplas. Isto é feito por meio da cláusula group by (onde os seus atributos são utilizados para formar os grupos).

Em outros casos, pode ser mais interessante definir condições e aplicá-las a grupos do que aplicá-las a tuplas. Isto é feito pela cláusula having. Como os predicados da cláusula são aplicados após a formação dos grupos, funções agregadas podem ser usadas.

select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P
group by $A_1$
having <função agregada>;

### 8.1.6 Subconsultas Aninhadas

Uma expressão select-from-where aninhada dentro de outra consulta é considerada uma subconsulta.

Aplicações:
– Membros de Conjuntos;
Verificar se uma tupla é membro ou não de uma relação. O conectivo **in** testa os membros de um conjunto, no qual este conjunto é a coleção de valores produzidos na clásula select.

select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P in (select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P);

– Comparações de Conjuntos;

Permite usar comparações do tipo > some (maior que ao menos uma), <= some, = some, etc... ou > all (maior que todas), >= all, etc.

select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P > all (select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P);

– Verificação de relações vazias

Pode-se testar se o resultado de uma subconsulta é vazio. O construtor exists retorna o valor true se o argumento da subconsulta é não-vazio.

select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P exists (select $A_1$, $A_2$, ..., $A_n$
from $r_1$, $r_2$, ..., $r_n$
where P);

### 8.1.7 Visões

Uma visão é qualquer relação que não faz parte do modelo lógico do banco de dados, mas que é visível ao usuário, como uma relação virtual.

O conjunto de tuplas de uma relação visão é resultado de uma expressão de consulta que foi definido no momento de sua execução. Logo, se uma relação visão é computada e armazenada, esta pode tornar-se desatualizada se as relações usadas em sua geração sofrerem modificações.

Quando uma visão é definida, o sistema de banco de dados armazena sua definição ao invés do resultado da expressão SQL que a definiu. Sempre que a relação visão é usada, ela é sobreposta pela expressão da consulta armazenada, de maneira que, sempre que a consulta for solicitada, a relação visão será recomputada.

Alguns sistemas de banco de dados permitem que as relações de visões sejam materializadas, garantindo que se ocorrerem modificações nas relações reais usadas na definição da visão, também a visão será modificada. Contudo, esta abordagem pode incorrer em custos de armazenamento e atualizações de sistema.

As visões em SQL são geradas a partir do comando create view. A cláusula padrão é:

CREATE VIEW <nome da visão> AS <expressão de consulta>;

Caso não necessitemos mais de uma dada visão, podemos eliminá-la por meio do comando:

DROP VIEW <nome da visão>;

Modificações no Banco de Dados por meio de SQL

### 8.1.8 Inserção

O comando INSERT é utilizado para adicionar uma única tupla a uma relação. Para inserirmos dados em uma relação devemos especificar a relação e uma lista de valores para a tupla a ser inserida.
Obs.: Os valores devem ser inseridos na mesma ordem na qual os atributos correspondentes foram especificados na criação da tabela de dados.

A cláusula padrão é:

INSERT INTO <relação>
VALUES <lista com valores dos atributos>

### 8.1.9 Atualização

O comando UPDATE é utilizado para modificar valores de atributos de uma ou mais tuplas selecionadas. As atualizações servem para modificar valores de tuplas que estão inseridas em um certo critério. Para tal usa-se o comando update. A cláusula padrão é:

UPDATE <relação>
SET <lista dos atributos a serem alterados e seus respectivos valores>
WHERE <condição>

### 8.1.10 Remoção

O comando DELETE remove tuplas de uma relação. A remoção de valores num banco de dados consiste na exclusão de tuplas que satisfaçam certa condição especificada na cláusula WHERE. As tuplas são explicitamente excluídas de uma tabela a cada vez.

A expressão padrão é:

DELETE FROM r
WHERE P

### 8.1.11 SQL DDL

Por meio da SQL DDL pode ser especificado não apenas o conjunto de relações, mas também informações sobre cada uma das relações existentes no banco de dados:

- O esquema de cada relação.
- O domínio dos valores associados a cada atributo.
- As regras de integridade.
- O conjunto de índices para manutenção de cada relação.
- Informações sobre segurança e autoridade sobre cada relação.
- A estrutura de armazenamento físico de cada relação no disco.

**Exemplos de Consultas SQL Básicas:**

A forma básica do comando SELECT, algumas vezes chamada de mapping ou bloco SELECT FROM WHERE, é formada por três cláusulas - SELECT, FROM e WHERE:

Consulta 0: Recuperar a data de nascimento e o endereço do empregado cujo nome é 'John B. Smith'.

```
Q0:     SELECT DATANASC, ENDEREÇO
        FROM EMPREGADO
        WHERE PNOME = 'John' AND MNOME = 'B' AND SNOME = 'Smith'
```

Esta consulta envolve apenas a relação EMPREGADO declarada na cláusula FROM. A consulta seleciona as tuplas de EMPREGADO que satisfazem a condição da cláusula WHERE, e então projeta o resultado sobre os atributos listados na cláusula SELECT. Q0 é similar à expressão da álgebra relacional:

$\pi_{\text{DATANASC, ENDEREÇO}}$ ($\sigma_{\text{PNOME = 'John' AND MNOME = 'B' AND SNOME='Smith'}}$ (EMPREGADO))

Assim, uma simples consulta SQL com uma única relação é similar ao par SELECT-PROJECT. A única diferença da consulta SQL é que, na relação resultante, as tuplas podem estar duplicadas, exceto se tiver o operador DISTINCT.

**Consulta 1**: Encontrar o nome e o endereço de todos os empregados que trabalham para o departamento 'Pesquisa'.

Q1:
```
SELECT PNOME, SNOME, ENDEREÇO
FROM EMPREGADO, DEPARTAMENTO
WHERE DNOME = 'Pesquisa' AND DNUMERO = NDEP
```

Esta consulta é similar à seqüência SELECT-PROJECT-JOIN da álgebra relacional. Na cláusula WHERE a condição DNOME='Pesquisa' é uma **condição de seleção** e corresponde à operação SELECT álgebra relacional. A condição DNUMERO = NDEP é uma **condição join**, que corresponde à condição da operação JOIN.

**Consulta 2**: Para todo projeto localizado em 'Stafford', listar o número do projeto, o número do departamento responsável, e o sobrenome, endereço e data de nascimento do gerente responsável pelo departamento.

Q2:
```
SELECT PNUMERO, DNUM, SNOME, ENDEREÇO, DATANASC
FROM PROJETO, DEPARTAMENTO, EMPREGADO
WHERE DNUM =  DNUMERO AND SSNGER = NSS AND
              PLOCALIZAÇÃO = 'Stafford'
```

**Consulta 3**: Encontrar os nomes de empregados que trabalham em todos os projetos controlados pelo departamento 5.

Q3:
```
SELECT PNOME, SNOME
FROM EMPREGADO
WHERE   ((SELECT PNRO
          FROM TRABALHA_EM
          WHERE NSS = NSSEMP)
          CONTAINS
         (SELECT PNUMERO
          FROM PROJETO
          WHERE DNUM = 5))
```

O operador CONTAINS compara dois conjuntos de valores e retorna TRUE se um conjunto contém todos os valores do outro. A segunda consulta dentro dos parênteses recupera todos os números de projetos controlados pelo departamento 5. Para cada tupla de EMPREGADO, a primeira consulta dentro dos parênteses recupera os números de projetos sobre os quais os empregados trabalham. Se o resultado desta consulta contiver todos os projetos controlados pelo departamento 5, a tupla de EMPREGADO é selecionada e o nome deste empregado é recuperado. Note que a operação de comparação CONTAINS é similar em sua função à operação DIVISION da álgebra relacional.

**Consulta 4**: Fazer uma lista de números de projetos no qual um empregado, cujo sobrenome é 'Smith' , trabalha no projeto ou é gerente do departamento que controla o projeto.

Q4:
```
(SELECT PNUMERO
FROM PROJETO, DEPARTAMENTO, EMPREGADO
WHERE DNUM=DNUMERO AND NSSGER=NSS AND SNOME='Smith')
UNION
(SELECT PNUMERO
```

```
FROM PROJETO, TRABALHA_EM, EMPREGADO
WHERE PNUMERO=PNRO AND NSSEMP=NSS AND SNOME='Smith')
```

**Consulta 5**: Listar os nomes de todos os empregados com dois ou mais dependentes.

Q5:

```
SELECT SNOME, PNOME
FROM EMPREGADO
WHERE (  SELECT COUNT (*)
FROM DEPENDENTE
WHERE NSS=NSSEMP) >= 2
```

onde (*) indica o número de tuplas.

**Consulta 6**: Listar os nomes dos empregados que não possuem dependentes.

Q6:
```
SELECT PNOME, SNOME
FROM EMPREGADO
WHERE NOT EXISTS (    SELECT *
                      FROM DEPENDENTE
                      WHERE NSS=NSSEMP)
```
onde * indica todos os valores de atributos das tuplas selecionadas.

**Consulta 7**: Listar os nomes dos gerentes que têm ao menos um dependente.

Q7:
```
SELECT PNOME, SNOME
FROM EMPREGADO
WHERE   EXISTS (      SELECT *
                      FROM DEPENDENTE
                      WHERE NSS=NSSEMP)
        AND
        EXISTS (      SELECT *
                      FROM DEPARTAMENTO
                      WHERE NSS=NSSGER)
```

# 9 Dependências Funcionais e Normalização de Base de Dados Relacionais

## 9.1 Diretrizes para o Projeto Informal de Esquemas de Relações

Nessa seção serão discutidas métricas de qualidade informal para projeto de esquemas de relações, quais sejam:

- Semântica de atributos.
- Redução de valores redundantes em tuplas.
- Redução de valores nulos em tuplas.
- Não permissão de tuplas espúrias.

Essas métricas não são sempre independentes uma das outras, como será visto.

### 9.1.1 Semântica de atributos de relação

Assume-se que certo significado esteja associado aos atributos, para todo agrupamento de atributos que formam uma relação esquema. Intuitivamente, verifica-se que cada relação pode ser interpretada como um conjunto de fatos ou declarações. Este significado, ou semântica, especifica como podem ser interpretados os valores de atributos armazenados em uma tupla da relação, em outras palavras, como os valores de atributos estão relacionados uns com os outros. Em geral, é mais simples descrever a semântica de relações, ao invés da semântica de atributos de uma relação.

Para ilustrar, considere a versão simplificada da base de dados COMPANHIA da Figura 5.4. O significado do esquema da relação é simples - cada tupla representa um empregado, com valores para nome, número do seguro social, data de aniversário, endereço, e o número do departamento em que cada empregado trabalha. Além desses atributos existem aqueles atributos que são utilizados para estabelecer um relacionamento entre relações.

Assim, todas os esquemas de relações da Figura 5.4. podem ser considerados como um bom ponto de partida para manter clara a semântica. Pode-se assim, estabelecer as seguintes diretrizes para projetar esquemas de relações:

**Diretriz 1ª**: Projetar um esquema de relação de maneira que seja simples descrever seu significado. Normalmente, isso significa que não se pode combinar atributos de múltiplos tipos de entidades e tipos de relacionamentos numa simples relação. Intuitivamente, se um esquema de relação corresponde a um tipo de entidade ou tipo de relacionamento, o significado tende a ser claro. Por outro lado, tende ser uma mistura de múltiplas entidades e relacionamentos e, assim, semanticamente não-clara.

A relação esquema da Figura 9.1 a e b possuem semântica clara. Uma tupla na relação esquema EMP_DEPT da Figura 9.1a representa um mero empregado, mas inclui informações adicionais, tais como o nome do departamento em que o empregado trabalha (DNOME) e o número do seguro social do gerente do departamento (NSSGER). Na relação esquema EMP_PROJ da Figura 9.1b, cada tupla relaciona um empregado a um projeto, mas também, incluem nome do empregado (ENOME), nome do projeto (PNOME) e a localização do projeto (PLOCALIZAÇÃO). Embora não exista, logicamente, nada de errado com esses esquemas, eles são considerados um projeto pobre, pois viola a Diretriz 1 porque *mistura atributos de entidades distintas do mundo real*; EMP_DEPT mistura atributos de empregados e departamentos, e EMP_PROJ mistura atributos de empregados e projetos. Eles podem ser usados como visões, mas podem causar problemas quando usados como relações da base de dados, como será discutido mais adiante.

(a) EMP_DEPT

| ENOME | NSS | DATANASC | ENDEREÇO | DNUMERO | DNOME | NSSGER |
|---|---|---|---|---|---|---|

(b) EMP_PROJ

| NSS | PNUMERO | HORAS | ENOME | PNOME | PLOCALIZAÇÃO |
|---|---|---|---|---|---|

**Figura 9.1**

## Informação redundante em tuplas e anomalias de atualizações

Uma das metas do projeto de esquemas é a minimização do espaço de armazenamento que relações da base ocupam. O agrupamento de atributos em esquemas de relações tem um efeito significativo no espaço de armazenamento. Por exemplo, compare o espaço usado pelas duas relações base EMPREGADO e DEPARTAMENTO na Figura 9.2 com o espaço utilizado por EMP_DEPT na relação base da Figura 9.3, que é o resultado da aplicação do JOIN NATURAL entre EMPREGADO e DEPARTAMENTO.

**EMPREGADO**

| ENOME | NSS | DATANASC | ENDEREÇO | DNUMERO |
|---|---|---|---|---|
| John Smith | 123456789 | 09-JAN-55 | R. A, 1 | 5 |
| Franklin Wong | 333445555 | 08-DEZ-45 | R. B, 2 | 5 |
| Alícia Zelaya | 999887777 | 19-JUL-58 | Av. C, 3 | 4 |
| Jennifer Wallace | 987654321 | 20-JUN-31 | Trav. D, 4 | 4 |
| Ramesh Narayan | 666884444 | 15-SET-52 | R. E, 5 | 5 |
| Joyce English | 453453453 | 31-JUL-62 | R. F, 6 | 5 |
| Ahmad Jabbar | 987987987 | 29-MAR-59 | Av G, 7 | 4 |
| James Borg | 888665555 | 10-NOV-27 | Av H, 8 | 1 |

**DEPARTAMENTO**

| DNOME | DNUMERO | NSSGER |
|---|---|---|
| Pesquisa | 5 | 333445555 |
| Administrativo | 4 | 987654321 |
| Gerencial | 1 | 888665555 |

**LOCAIS_DEPTO**

| DNUMERO | DLOCALIZAÇÃO |
|---|---|
| 1 | Houston |
| 4 | Stafford |
| 5 | Bellaire |
| 5 | Sugariand |
| 5 | Houston |

**TRABALHA_EM**

| NSSEMP | PNRO | HORAS |
|---|---|---|
| 123456789 | 1 | 32.5 |
| 123456789 | 2 | 7.5 |
| 666884444 | 3 | 40.0 |
| 453453453 | 1 | 20.0 |
| 453453453 | 2 | 20.0 |
| 333445555 | 2 | 10.0 |
| 333445555 | 3 | 10.0 |
| 333445555 | 10 | 10.0 |
| 333445555 | 20 | 10.0 |
| 999887777 | 30 | 30.0 |
| 999887777 | 10 | 10.0 |
| 987987987 | 10 | 35.0 |
| 987987987 | 30 | 5.0 |
| 987654321 | 30 | 20.0 |
| 987654321 | 20 | 15.0 |
| 888775555 | 20 | null |

**PROJETO**

| PNOME | PNUMERO | PLOCALIZAÇÃO | DNUM |
|---|---|---|---|
| ProdutoX | 1 | Bellaire | 5 |
| ProdutoY | 2 | Sugarland | 5 |
| ProdutoZ | 3 | Houston | 5 |
| Automação | 10 | Stafford | 4 |
| Reorganização | 20 | Houston | 1 |
| Beneficiamento | 30 | Stafford | 4 |

**Figura 9.2**

EMP_DEPT

| ENOME | NSS | DATANASC | ENDEREÇO | DNUMERO | DNOME | NSSGER |
|---|---|---|---|---|---|---|
| John Smith | 123456789 | 09-JAN-55 | R. A, 1 | 5 | Pesquisa | 333445555 |
| Franklin Wong | 333445555 | 08-DEZ-45 | R. B, 2 | 5 | Pesquisa | 333445555 |
| Alícia Zelaya | 999887777 | 19-JUL-58 | Av. C, 3 | 4 | Administrativo | 987654321 |
| Jennifer Wallace | 987654321 | 20-JUN-31 | Trav. D, 4 | 4 | Administrativo | 987654321 |
| Ramesh Narayan | 666884444 | 15-SET-52 | R. E, 5 | 5 | Pesquisa | 333445555 |
| Joyce English | 453453453 | 31-JUL-62 | R. F, 6 | 5 | Pesquisa | 333445555 |
| Ahmad Jabbar | 987987987 | 29-MAR-59 | Av G, 7 | 4 | Administrativo | 987654321 |
| James Borg | 888665555 | 10-NOV-27 | Av H, 8 | 1 | Gerencial | 888665555 |

EMP_PROJ

| NSS | PNUMERO | HORAS | ENOME | PNOME | PLOCALIZAÇÃO |
|---|---|---|---|---|---|
| 123456789 | 1 | 32.5 | John Smith | ProdutoX | Bellaire |
| 123456789 | 2 | 7.5 | John Smith | ProdutoY | Sugarland |
| 666884444 | 3 | 40.0 | Ramesh Narayan | ProdutoZ | Houston |
| 453453453 | 1 | 20.0 | Joyce English | ProdutoX | Bellaire |
| 453453453 | 2 | 20.0 | Joyce English | ProdutoY | Sugarland |
| 333445555 | 2 | 10.0 | Franklin Wong | ProdutoY | Sugarland |
| 333445555 | 3 | 10.0 | Franklin Wong | ProdutoZ | Houston |
| 333445555 | 10 | 10.0 | Franklin Wong | Automação | Stafford |
| 333445555 | 20 | 10.0 | Franklin Wong | Reorganização | Houston |
| 999887777 | 30 | 30.0 | Alícia Zelaya | Beneficiamento | Stafford |

| | | | | | |
|---|---|---|---|---|---|
| 999887777 | 10 | 10.0 | Alícia Zelaya | Automação | Stafford |
| 987987987 | 10 | 35.0 | Ahmad Jabbar | Automação | Stafford |
| 987987987 | 30 | 5.0 | Ahmad Jabbar | Beneficiamento | Stafford |
| 987987987 | 30 | 20.0 | Jennifer Wallace | Beneficiamento | Stafford |
| 987987987 | 20 | 15.0 | Jennifer Wallace | Reorganização | Houston |
| 888665555 | 20 | null | James Borg | Reorganização | Houston |

**Figura 9.3**

Em EMP_DEPT, os valores de atributos pertencentes a um particular departamento (DNUMERO, DNOME, NSSGER) estão repetidos para todos os empregados que trabalham para um departamento. Em contraste, as informações de departamento aparecem apenas uma vez para cada departamento na relação DEPARTAMENTO da Figura 9.2, e apenas o número do departamento (DNUMERO) é repetido na relação EMPREGADO para cada empregado que trabalha no departamento. Similarmente, o mesmo comentário se aplica para a relação EMP_PROJ.

Um outro problema sério identificado, quando se utilizam relações similares às encontradas na Figura 9.3 como relações base, é o problema da anomalia de alterações. Podem ser classificadas em anomalias de inserção, remoção e modificações.

## 9.2 Anomalia de inserção

Pode-se diferenciar dois tipos de anomalias de inserção.

- Para inserir uma nova tupla empregado em EMP_DEPT, deve-se incluir valores de atributos do departamento em que o empregado trabalha, ou *null* se o empregado ainda não pertencer a um departamento. Por exemplo, para inserir uma nova tupla para o empregado que trabalha no departamento 5, deve-se tomar cuidado em assegurar que os valores de atributos sejam inseridos corretamente, tal que fiquem consistentes com os valores do departamento 5 em outras tuplas de EMP_DEPT. No projeto ilustrado na Figura 9.2 esse cuidado não precisa ser tomado, uma vez que apenas o número do departamento é necessário em cada tupla de empregado; os outros valores de atributos do departamento 5 são registrados apenas uma vez na base de dados, como uma simples tupla na relação DEPARTAMENTO.
- É difícil inserir um novo departamento que ainda não tenha empregados na relação EMP_DEPT. A única maneira de fazer isso é colocar valores *null* em atributos de empregados. Isso provoca um problema pois NSS é a chave primária de EMP_DEPT, e supõe-se que cada tupla representa uma entidade empregado — não uma entidade departamento. Mais do que isso, quando o primeiro empregado for associado a esse departamento, não será mais necessário a tupla com valores *null*. Este problema não ocorre no projeto da Figura 9.2, porque um departamento é inserido na relação DEPARTAMENTO sem que nenhum empregado trabalhe nele, e quando um empregado for associado ao departamento, uma tupla correspondente é inserida em EMPREGADO.

### 9.2.1 Anomalia de remoção

Este problema está relacionado à segunda anomalia de inserção discutida acima. Se for removida uma tupla empregado de EMP_DEPT e esta tupla for à última de um particular departamento, a informação correspondente ao departamento será perdida. Este problema não ocorre na base de dados da Figura 9.2 pois as tuplas de DEPARTAMENTO são armazenadas separadamente das tuplas de EMPREGADO, somente o atributo DNÙMERO de EMPREGADO relaciona os dois.

### 9.2.2 Anomalia de modificação

Em EMP_DEPT, se for mudado o valor de um dos atributos de um departamento qualquer, por exemplo, o gerente do departamento 5, deve ser mudado as tuplas de todos os empregados que trabalham neste departamento; por outro lado, a base de dados se tornará inconsistente.

### 9.2.3 Discussão

De acordo com as anomalias acima, pode-se estabelecer a seguinte diretriz:

**Diretriz 2ª**: Projetar esquemas de relações de maneira que nenhuma anomalia de alteração ocorra em relações. Se existir alguma anomalia, isso deverá ser considerado para que as modificações pelos programas ocorram corretamente.

A segunda diretriz é consistente e reforça a primeira diretriz. Verifica-se, porém, que existe a necessidade de uma abordagem mais formal para auxiliar na avaliação de que projetos obedecem estas diretrizes. Nas seções seguintes isso será fornecido. É importante notar que estas diretrizes podem, algumas vezes, ter que ser violadas a fim de aumentar o desempenho de certas consultas. Por exemplo, se for importante recuperar, rapidamente, informações pertinentes ao departamento, através de atributos de um empregado que trabalhe no departamento, o esquema EMP_DEPT pode ser usado como uma relação base. No entanto, deve-se assegurar que as anomalias sejam bem conhecidas para que as modificações nesta relação base não a tornem inconsistente. Em geral é aconselhável utilizar relações base livre de anomalias e especificar visões utilizando JOIN´s para unir atributos freqüentemente referenciados em consultas importantes. Isso reduz o número de JOIN´s especificados em consultas, simplifica as consultas, e em alguns casos, eleva o desempenho.

### 9.2.4 Valores null em tuplas

Em alguns projetos de esquemas pode-se agrupar atributos em uma “grande” relação. Se muitos atributos não se aplicarem a todas as tuplas da relação, ter-se-á muitos valores *nulls* na relação. Isto pode despender espaço armazenamento e pode também trazer problemas de entendimento do significado dos atributos na especificação de operações JOIN´s. Um outro problema com valores *nulls* é que não se pode contá-los quando operações de agregação, tais como COUNT ou SUM são aplicadas. Mais que isso, os valores *nulls* podem ter múltiplas interpretações, tais como:

- O atributo pode não se aplicar a tupla.
- O valor de atributo para a tupla é desconhecido.
- O valor é conhecido, porém não foi registrado ainda.

Ter a mesma representação, *null*, para diferentes significados pode comprometer as consultas. Assim, pode-se estabelecer uma outra diretriz:

**Diretriz 3ª**: Tanto quanto possível, evite colocar atributos em um esquema de relação base cujos valores possam ser *null*. Se forem inevitáveis os valores *nulls*, esteja seguro que eles se apliquem apenas em casos excepcionais e não se apliquem na maioria das tuplas da relação.

Por exemplo, se apenas 10% dos empregados tiverem secretárias, não existe razão para incluir um atributo NÚMERO_SECRETÁRIA na relação EMPREGADO; ao invés disso, uma relação EMP_SEC(ESSN, NÚMERO_SECRETÁRIA) poderia ser criada para conter apenas as tuplas indicando o ESSN de empregados que possuem secretárias.

### 9.2.5 Tuplas espúrias

Considere os esquemas de relação EMP_LOCS e EMP_PROJ1 da Figura 9.4a, que pode substituir a relação EMP_PROJ da Figura 9.1b.

(a)

EMP_LOCS

| ENAME | PLOCALIZAÇÃO |
|---|---|

EMP_PROJ1

| NSS | PNUMERO | HORAS | PNOME | PLOCALIZAÇÃO |
|---|---|---|---|---|

(b)

EMP_LOCS

| ENAME | PLOCALIZAÇÃO |
|---|---|
| John Smith | Bellaire |
| John Smith | Sugarland |

| | |
|---|---|
| Ramesh Narayan | Houston |
| Joyce English | Bellaire |
| Joyce English | Sugarland |
| Franklin Wong | Sugarland |
| Franklin Wong | Houston |
| Franklin Wong | Stafford |
| Alícia Zelaya | Stafford |
| Ahmad Jabbar | Stafford |
| Jennifer Wallace | Stafford |
| Jennifer Wallace | Houston |
| James Borg | Houston |

EMP_PROJ1

| NSS | PNUMERO | HORAS | PNOME | PLOCALIZAÇÃO |
|---|---|---|---|---|
| 123456789 | 1 | 32.5 | ProdutoX | Bellaire |
| 123456789 | 2 | 7.5 | ProdutoY | Sugarland |
| 666884444 | 3 | 40.0 | ProdutoZ | Houston |
| 453453453 | 1 | 20.0 | ProdutoX | Bellaire |
| 453453453 | 2 | 20.0 | ProdutoY | Sugarland |
| 333445555 | 2 | 10.0 | ProdutoY | Sugarland |
| 333445555 | 3 | 10.0 | ProdutoZ | Houston |
| 333445555 | 10 | 10.0 | Automação | Stafford |
| 333445555 | 20 | 10.0 | Reorganização | Houston |
| 999887777 | 30 | 30.0 | Beneficiamento | Stafford |
| 999887777 | 10 | 10.0 | Automação | Stafford |
| 987987987 | 10 | 35.0 | Automação | Stafford |
| 987987987 | 30 | 5.0 | Beneficiamento | Stafford |
| 987987987 | 30 | 20.0 | Beneficiamento | Stafford |
| 987987987 | 20 | 15.0 | Reorganização | Houston |
| 888665555 | 20 | null | Reorganização | Houston |

## Figura 9.4

Uma tupla em EMP_LOCS significa que o empregado cujo nome é ENOME trabalha em algum projeto cuja localização é PLOCALIZAÇÂO. Uma tupla em EMP_PROJ1 significa que o empregado cujo número do seguro social é NSS trabalha HORAS por semana em um projeto cujo nome, número e localização são PNOME, PNUMERO e PLOCALIZAÇÂO. A Figura 9.4b mostra a extensão de EMP_LOCS e EMP_PROJ1 correspondente à relação extensão EMP_PROJ da Figura 9.3, aplicando-se operações de projeção (¶) adequadas.

Suponha agora que EMP_PROJ1 e EMP_LOCS sejam utilizadas como relações base ao invés de EMP_PROJ. Isto seria, particularmente, um projeto ruim, pois não se pode recuperar as informações que existiam originalmente em EMP_PROJ a partir de EMP_PROJ1 e EMP_LOCS. Se uma operação JOIN-NATURAL for aplicada em EMP_PROJ1 e EMP_LOCS, surgirão mais tuplas que existiam em EMP_PROJ. A Figura 9.5 ilustra o resultado obtido aplicando-se o join, considerando apenas as tuplas existentes acima da linha pontilhada, para reduzir o tamanho da relação resultante. As tuplas adicionais são chamadas **tuplas espúrias** porque elas representam informações espúrias ou erradas, e por isso elas não válidas. As tuplas espúrias estão marcadas por asteriscos (*) na Figura 9.5.

A decomposição de EMP_PROJ em EMP_PROJ1 e EMP_LOCS é ruim porque quando é aplicada a operação JOIN-NATURAL, não são obtidas as informações originais corretas. Isto porque foi escolhido PLOCALIZAÇÃO como o atributo que relaciona EMP_LOCS e EMP_PROJ1, e PLOCALIZAÇÃO não é nem uma chave-primária e nem uma chave-estrangeira. Pode-se agora estabelecer uma outra diretriz para projeto:

**Diretriz 4ª**: Projetar esquemas de relações tal que, quando aplicadas operações JOIN-NATURAIS, os atributos nas condições-joins envolvam atributos que sejam ou chaves-primárias ou chaves-estrangeiras de maneira a garantir que nenhuma tupla espúria seja gerada.

| | NSS | PNUMERO | HORAS | PNOME | PLOCALIZAÇÃO | ENAME |
|---|---|---|---|---|---|---|
| | 123456789 | 1 | 32.5 | ProdutoX | Bellaire | John Smith |
| * | 123456789 | 1 | 32.5 | ProdutoX | Bellaire | Joyce English |
| | 123456789 | 2 | 7.5 | ProdutoY | Sugarland | John Smith |
| * | 123456789 | 2 | 7.5 | ProdutoY | Sugarland | Joyce English |
| * | 123456789 | 2 | 7.5 | ProdutoY | Sugarland | Franklin Wong |
| | 666884444 | 3 | 40.0 | ProdutoZ | Houston | Ramesh Narayan |
| * | 666884444 | 3 | 40.0 | ProdutoZ | Houston | Franklin Wong |
| * | 453453453 | 1 | 20.0 | ProdutoX | Bellaire | John Smith |

| | | | | | | |
|---|---|---|---|---|---|---|
| | 453453453 | 1 | 20.0 | ProdutoX | Bellaire | Joyce English |
| * | 453453453 | 2 | 20.0 | ProdutoY | Sugarland | John Smith |
| | 453453453 | 2 | 20.0 | ProdutoY | Sugarland | Joyce English |
| * | 453453453 | 2 | 20.0 | ProdutoY | Sugarland | Franklin Wong |
| * | 333445555 | 2 | 10.0 | ProdutoY | Sugarland | John Smith |
| * | 333445555 | 2 | 10.0 | ProdutoY | Sugarland | Joyce English |
| | 333445555 | 2 | 10.0 | ProdutoY | Sugarland | Franklin Wong |
| * | 333445555 | 3 | 10.0 | ProdutoZ | Houston | Ramesh Narayan |
| | 333445555 | 3 | 10.0 | ProdutoZ | Houston | Franklin Wong |
| | 333445555 | 10 | 10.0 | Automação | Stafford | Franklin Wong |
| * | 333445555 | 20 | 10.0 | Reorganização | Houston | Ramesh Narayan |
| | 333445555 | 20 | 10.0 | Reorganização | Houston | Franklin Wong |

**Figura 9.5**

Obviamente, estas diretrizes informais necessitam ser estabelecidas de maneira mais formal.

## 9.3 Dependências Funcionais

Um conceito simples, porém, muito importante em projetos de esquemas relacionais é o de dependência funcional. Nesta seção será definido formalmente este conceito, e na seção 9.2.2 será verificado como trabalhar com esquemas de relações em formas normais.

### 9.3.1 Definição de Dependência Funcional

Dependências Funcionais são restrições ao conjunto de relações válidas. Elas permitem expressar determinados fatos em banco de dados relativos ao empreendimento que se deseja modelar. Anteriormente foi definido o conceito de superchave. Seja *R* o esquema de uma relação. Um subconjunto *K* de *R* é uma *superchave* de *R* em qualquer relação válida *r(R)* para todos os pares $t_1$ e $t_2$ de tuplas em *r* tal que $t_1 \neq t_2$, então $t_1[K] \neq t_2[K]$. Isto é, nenhum par de tuplas em qualquer relação válida *r(R)* deve ter o mesmo valor no conjunto de atributos K.
A noção de dependência funcional generaliza a noção de superchave. Seja $\alpha \subseteq R$ e $\beta \subseteq R$. A dependência funcional :

$$\alpha \rightarrow \beta$$

realiza-se em *R* se, em qualquer relação válida *r(R)*, para todos os pares de tuplas $t_1$ e $t_2$ em *r* tal que $t_1[\alpha] = t_2[\alpha]$, $t_1[\beta] = t_2[\beta]$ também será verdade.
Usando a notação de dependência funcional, dizemos que K é uma superchave de R se $K \rightarrow R$. Isto é, K é uma superchave se, para todo $t_1[K] = t_2[K]$, $t_1[R] = t_2[R]$ (isto é $t_1 = t_2$). A dependência funcional permite expressar restrições que as superchaves não expressam. Considere o esquema:

*Esquema_info_empréstimo = (nome_agência, número_empréstimo, nome_cliente, total)*

O conjunto de dependências funcional que queremos garantir para esse esquema de relação é:

*número_empréstimo→total*
*número_empréstimo→nome_agência*

Entretanto, não se espera realizar dependência funcional para:

*número_empréstimo→nome_cliente*

já que, em geral, um empréstimo pode ser contraído por mais de um cliente (por exemplo para ambos os membros de um casal, marido-mulher).
Podemos usar dependência funcional de dois modos:

1. Usando-as para o estabelecimento de restrições sobre um conjunto de relações válidas. Deve-se assim, concentrá-las *somente* àquelas que devem satisfazer um dado conjunto de dependências funcionais. Se desejasse restringi-las a relações do

esquema em *R* que satisfaçam o conjunto *F* de dependências funcionais, diz-se que *F realiza-se* em *R*.

2. Usando-as para verificações de relações, de modo a saber se as últimas são válidas sob um conjunto de dependências funcionais. Se uma relação *r* é legal sobre um conjunto *F* de dependências funcionais, diz-se que *r satisfaz F*.

Considera-se a relação *r* mostrada abaixo para verificar quais dependências funcionais são satisfeitas. Observa-se que $A \rightarrow C$ é satisfeita. Duas tuplas têm valor $a_1$ em *A.* Essas tuplas têm o mesmo valor de *C* – denominado $c_1$. De modo similar, duas tuplas com valor $a_2$ em *A* têm o mesmo valor $c_2$ em *C*. Não existe outro par de tuplas distintas que tenha, em *A*, os mesmos valores. A dependência funcional $C \rightarrow A$, entretanto não é satisfeita. Para confirmar esta afirmação, considere as tuplas $t_1 = (a_2, b_2, c_2, d_3)$ e $t_2 = (a_3, b_3, c_2, d_4)$. Essas tuplas têm os mesmos valores $c_2$ em *C*, mas elas possuem valores diferentes em *A*, $a_2$ e $a_3$, respectivamente. Assim, encontra-se um par de tuplas $t_1$ e $t_2$ tal que $t_1[C] = t_2[C]$, mas $t_1[A] \neq t_2[A]$.

| A | B | C | D |
|---|---|---|---|
| $a_1$ | $b_1$ | $c_1$ | $d_1$ |
| $a_1$ | $b_2$ | $c_1$ | $d_2$ |
| $a_2$ | $b_2$ | $c_2$ | $d_2$ |
| $a_2$ | $b_3$ | $c_2$ | $d_3$ |
| $a_3$ | $b_3$ | $c_2$ | $d_4$ |

**Figura 9.6 - Relação *r* de exemplo**

Uma dependência funcional é uma restrição entre dois conjuntos de atributos da base de dados. Considere-se que o esquema da base de dados relacional tenha n atributos $A_1$, $A_2$, ..., $A_n$ ; e que toda a base de dados seja descrita por um único esquema de relação **universal** R = { $A_1$, $A_2$, ..., $A_n$ }. Isso não significa que isso de fato deverá acontecer; esta suposição é feita apenas para desenvolver uma teoria formal sobre dependência de dados.

Uma **dependência funcional**, denotada por X→Y, entre dois conjuntos de atributos X e Y que são subconjuntos de R, especifica uma restrição sobre as possíveis tuplas que podem existir na relação instância r de R. A restrição estabelece que para quaisquer tuplas $t_1$ e $t_2$ em r, se $t_1[X]= t_2[X]$, então $t_1[Y]= t_2[Y]$. Isto significa que os valores da componente Y das tuplas em r dependem, ou são determinados pelos valores da componente X ou, alternativamente, os valores da componente X de uma tupla **determinam** de maneira única (ou **funcionalmente**) os valores da componente Y. Pode-se também dizer que existe uma dependência funcional de X para Y ou que Y é **dependente funcionalmente** de X. A abreviatura de dependência funcional é **DF**.

Note que:

- Se uma restrição sobre R estabelece que não pode existir mais que uma tupla com um dado valor de X, em quaisquer instâncias de relação r(R) - isto é, X é uma chave-candidata de R - isto implica que X→Y para quaisquer subconjuntos de atributos Y de R.
- Se X→Y em R, isto não significa que Y→X em R.

Uma dependência funcional é uma propriedade do significado ou **semântica** dos atributos em um esquema de relação R. Utiliza-se o entendimento da semântica de atributos de R - isto é, como eles se relacionam - para especificar as dependências funcionais envolvidas em todas as instâncias da relação r (extensão) de R. As instâncias r que satisfazem as restrições de dependência funcional especificadas sobre atributos de R são chamadas **extensões legais**, pois obedecem as restrições de dependência funcional. Assim, a principal utilização das dependências funcionais é o de descrever um esquema de relação R especificando restrições sobre seus atributos que devem ser válidas todas às vezes.

Isto significa que uma dependência funcional é uma propriedade do esquema da relação (intenção) R e não de uma instância particular legal r (extensão) de R. Assim, uma DF não pode ser automaticamente inferida a partir de uma extensão de relação r, mas deve ser definida por alguém que conheça a semântica dos atributos de R. Por exemplo, na Figura 9.7, uma instância particular de um esquema de relação ENSINO é mostrada. Embora num primeiro momento possa parecer que TEXTO→CURSO, não se pode afirmar isso a menos que se conheça que isso seja verdade para todas as instâncias da relação ENSINO.

| PROFESSOR | CURSO | TEXTO |
|---|---|---|
| Smith | Estrutura de Dados | Bartram |
| Smith | Gerenciamento de dados | Al-Nour |
| Hall | Compiladores | Hoffman |
| Brown | Estrutura de Dados | Augenthaler |

**Figura 9.7**

## 9.3.2 Formas Normais Determinadas pelas Chaves Primárias

O processo de normalização foi proposto inicialmente por Codd em 1972. Esta é uma maneira mais formal de garantir as diretrizes descritas anteriormente.

### 9.3.2.1 Primeira Forma Normal (1FN)

A primeira forma normal é agora genericamente considerada como parte da definição formal de uma relação; historicamente foi definida para não permitir atributos multivalorados, compostos e suas combinações.

### 9.3.2.2 Segunda Forma Normal (2FN)

A segunda forma normal é baseada no conceito de dependência funcional total. Uma dependência funcional X→Y é uma dependência funcional total se, na remoção de qualquer atributo A de X significar que a dependência não mais existe; isto é, para todo atributo A ∈ X, (X-{A}) -x→Y. Por exemplo, na Figura 9.8b, {NSS, PNUMERO}→HORAS é uma dependência funcional total (nem NSS→HORAS e nem PNUMERO→HORAS são DF. Porém, a dependência {NSS, PNUMERO}→ENOME é parcial, pois NSS→ENOME.

Um esquema de relação R está na 2FN se qualquer atributo não-primo A de R for totalmente dependente funcionalmente de qualquer chave de R. Atributo não-primo é qualquer atributo que não seja membro de uma chave-candidata. A relação EMP_PROJ da Figura 9.8b está na 1FN, mas não na 2FN. O atributo não-primo ENOME viola a 2FN devido à df2, o mesmo acontece com os atributos não-primos PNOME e PLOCALIZAÇÃO em df3. As dependências funcionais df2 e df3 indicam a dependência parcial da chave-primária {NSS, PNUMERO}, violando assim a 2FN.

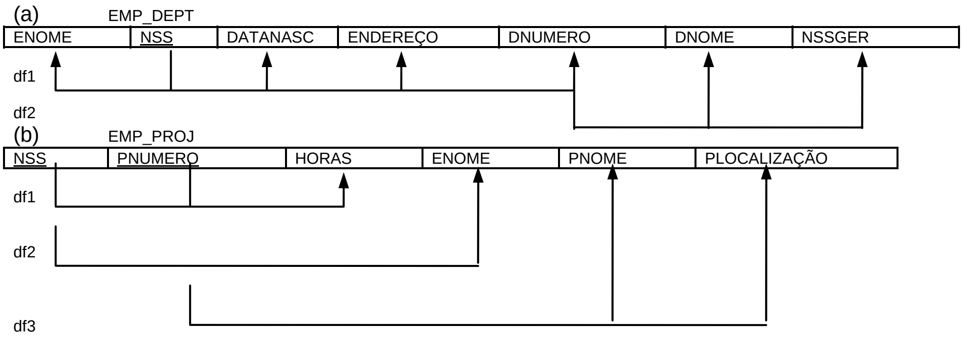



**Figura 9.8**

Se uma relação não está na 2FN, ela pode ser normalizada em um número de relações na 2FN. As dependências funcionais df1, df2 e df3 indicadas na Figura 9.9 podem ser consideradas para decompor a relação EMP_PROJ em 3 esquemas de relações EP1, EP2 e EP3, como mostra a Figura 9.9a. Cada um desses esquemas de relação satisfaz a 2FN. Verifica-se que as relações EP1, EP2 e EP3 evitam as anomalias que estava sujeita a relação EMP_PROJ.

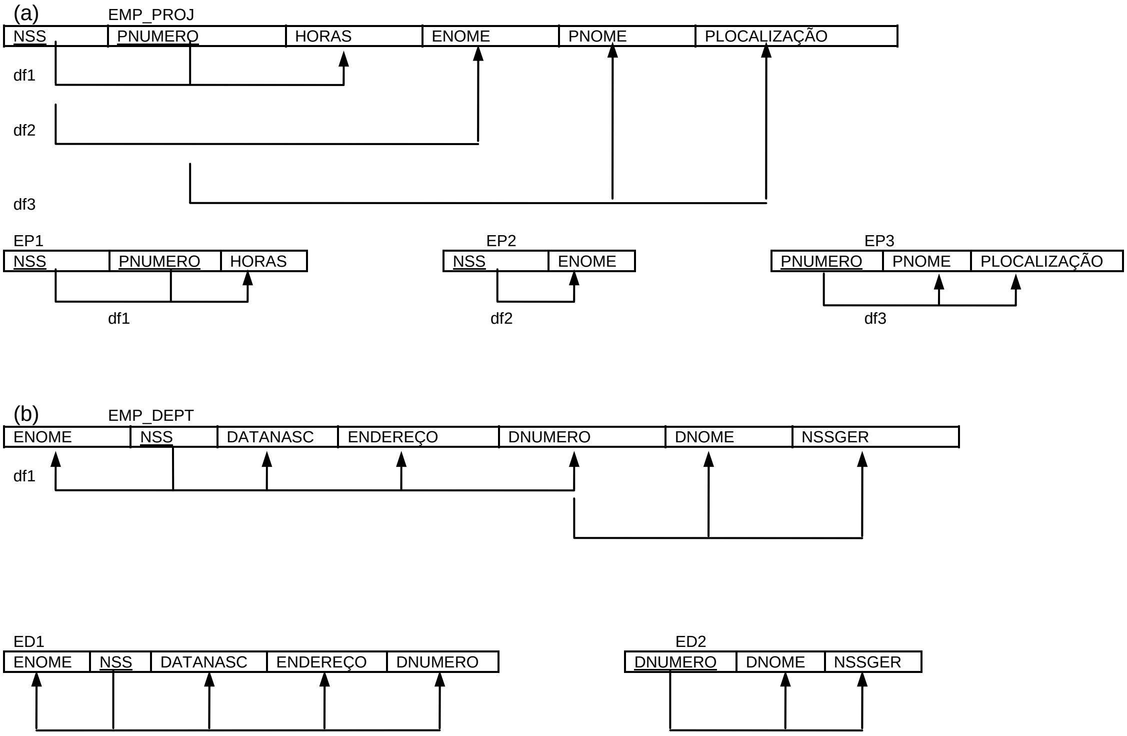



**Figura 9.9**

## 9.3.2.3 Terceira Forma Normal (3FN)

A terceira forma normal é baseada no conceito de dependência transitiva. Uma dependência X→Y em uma relação R é uma dependência transitiva se existir um conjunto de atributos Z que não é um subconjunto de qualquer chave de R, e tanto X→Z quanto Z→Y. Por exemplo, a dependência NSS→NSSGER é uma dependência transitiva em EMP_DEPT da Figura 9.9b. Diz-se que a dependência de NSSGER sobre o atributo chave NSS é transitiva via DNUMERO. Intuitivamente, verifica-se que a dependência de NSSGER sobre DNUMERO é indesejável uma EMP_DEPT desde que DNUMERO não é chave de EMP_DEPT.

Um esquema de relação R está na 3FN se ele estiver na 2FN e nenhum atributo não-primo de R é dependente transitivamente de qualquer chave de R. A relação EMP_DEPT da Figura 9.9b está na 2FN, pois não há dependência parcial de nenhum atributo não-primo sobre a chave. Porém, não está na 3FN, pois NSSGER e DNOME são dependentes transitivos de NSS via DNUMERO. Pode-se normalizar EMP_DEPT decompondo-o em dois esquemas de relação na 3FN, ED1 e ED2, como mostra a Figura 9.9b.

## *9.4 Definição Geral (2FN, 3FN, FNBC)*

Em geral desejamos projetar nossos esquemas de relações de modo que eles não tenham dependências parciais nem transitivas. Esses tipos de dependências causam as anomalias de atualização discutidas nas seções anteriores. Os passos para normalização para as relações na 3FN que discutimos até aqui proíbem dependências parciais e transitivas na chave primária. Essas definições, entretanto, não levam em conta outras chaves candidatas de uma relação, *se é que existem*!!

Entretanto nesta seção, trataremos de definições mais gerais de 2FN e 3FN para introduzirmos a Forma Normal de Boyce-Codd. Essas definições mais gerais não afetam a definição de 1FN. Utilizaremos aqui a definição de *atributo primo (principal)*, ou seja, um atributo integrante (parte) de qualquer chave candidata. Assim as dependências parcial e plenamente funcionais e dependências transitivas serão agora com relação a todas as chaves candidatas de uma relação.

### 9.4.2 Definição geral de segunda Forma Normal

Um esquema de relação R está na segunda forma normal (2FN) se todo atributo A em R não for parcialmente dependente de qualquer chave de R, ou em outras palavras, se todo atributo não-primo A em R for completamente dependente em termos funcionais de todas as chaves de R. Considere o esquema da relação LOTES mostrado na figura 9.10a, que descreve pedaços de terra para venda em vários municípios de um estado. Suponha que existam duas chaves candidatas: #ID_Propriedade e (Nome_Municipio, #LOTE); ou seja, números de lotes são únicos somente dentro de cada município, mas números de ID_PROPRIEDADE são únicos ao longo dos municípios em todo seu estado.

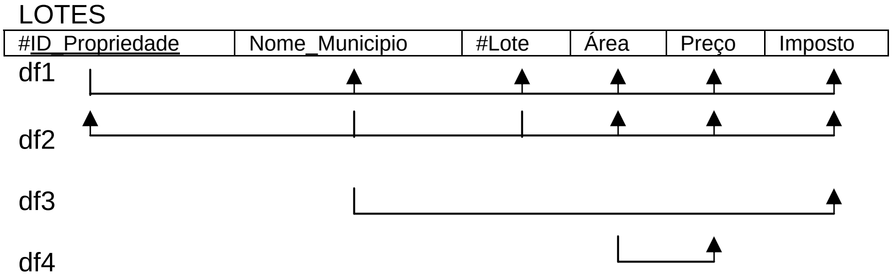
Com base nas duas chaves candidatas #ID_Propriedade e (Nome_Municipio, #LOTE), sabemos que as dependências funcionais DF1 e DF2 da figura 9.10a se mantêm. Escolhemos #ID_Propriedade como chave primária. Além disso, também supomos que existam mais duas dependências funcionais adicionais na relação LOTES:

- DF3: Nome_Municipio → Imposto
- DF4: Área → Preço

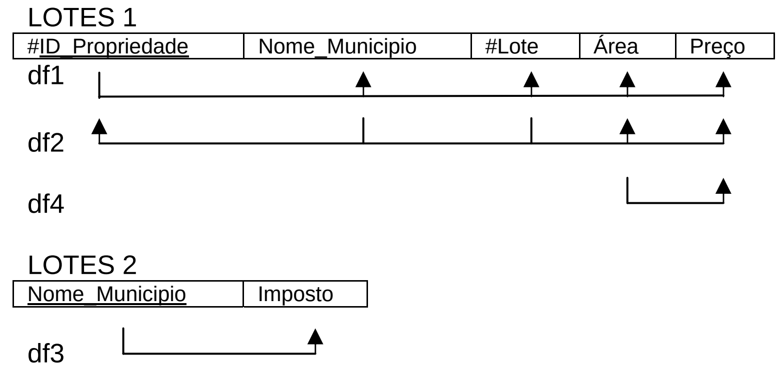
Traduzindo o significado das dependências funcionais temos que a DF3 diz que o imposto é fixo para um dado município (não varia lote por lote dentro do mesmo município). Enquanto DF4 diz que o preço de um lote é determinado por sua área, independentemente do município em que esteja localizado. O esquema da relação LOTES desrespeita a definição geral da 2FN porque IMPOSTO é parcialmente dependente da chave candidata (Nome_Municipio, #Lote), devido a DF3. Para normalizar LOTES para 2FN, decompomos o mesmo em duas relações LOTES1 e LOTES2, mostradas na figura 9.10b. Construímos LOTES1 removendo o atributo IMPOSTO que desrespeita a 2FN de LOTES e colocando-o com Nome_Municipio em uma outra relação LOTES2. Tanto LOTES1 como LOTES2 estão na 2FN. Note que DF4 não desrespeita a 2FN e é transportada para LOTES1.

(a) esquema relação lotes 1FN.

LOTES



(b) esquemas da relação lotes 2FN.




(c) esquemas da relação lotes 3FN.

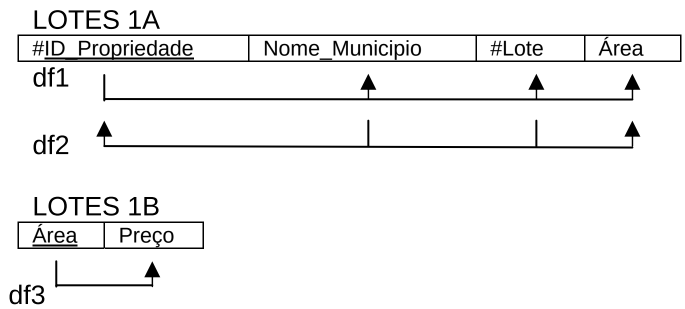



Figura 9.10 (a, b, c) - Normalização do esquema da relação LOTES.

## 9.4.3 Definição geral de terceira Forma Normal

Um esquema da relação R está na terceira forma normal (3FN) sempre que uma dependência funcional não trivial X→ A se mantiver em R, seja pelo motivo de que:

a) X é uma superchave de R, ou
b) A é um atributo primo de R.

De acordo com essa definição, LOTES2 está na 3FN. Entretanto, DF4 em LOTES1 viola a 3FN porque AREA não é um superchave e PREÇO não é o atributo principal (primo) de LOTES1.

Para normalizar LOTES1 para a 3FN, decompomos em LOTES1A e LOTES1B mostrados na figura 9.10c. Construímos LOTES1A retirando o atributo PREÇO que viola a 3FN de LOTES1 associando-o com AREA (o atributo do lado esquerdo de DF4 que causa a dependência transitiva) em uma outra relação denominada LOTES1B. Tanto LOTES1A como LOTES1B estão na 3FN. LOTES1 viola a 3FN por que PREÇO é transitivamente dependente de cada uma as chaves candidatas de LOTES1 através do atributo não-primo AREA.

## 9.4.4 Definição da Forma Normal de Boyce-Codd

A forma normal de Boyce-Codd (FNBC) foi proposta como uma forma mais simples que 3FN, mas foi considerada mais restrita na medida em que toda relação que está na FNBC também está na 3FN, entretanto o inverso não é verdadeiro.

A definição formal da FNBC difere ligeiramente da definição da 3FN. Um esquema da relação R está na FNBC se, sempre uma dependência funcional não-trivial X→ A se mantiver em R, X for uma superchave de R. A condição b da 3FN está ausente.

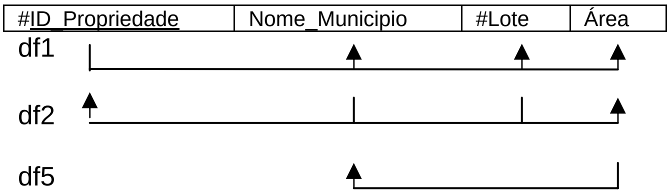
Preferencialmente o projeto de um banco de dados relacional deve empenhar-se em alcançar a FNBC ou a 3FN para todo esquema da relação. Alcançar o status de normalização somente na 1FN ou na 2FN não é considerado adequado, uma vez que essas formas foram desenvolvidas como caminho para 3FN e FNBC.

Para exemplificar a FNBC foi incluída uma DF5, em LOTES1A como ilustra a figura 9.11a. Junto com tal dependência funcional "adicionada", assumimos que existam milhares de lotes, mas apenas dois municípios X e Y. Supomos também que as áreas dos lotes do município X são 0,5; 06; ...; 1.0 alqueires e os lotes de Y são 1,1; 1,2; .....; 2,0 alqueires. Nesse caso teríamos DF5: AREA →Nome_Municipio.

A área de um lote que determina o município, conforme especificado por DF5, pode ser representada através de 16 tuplas numa relação em separado R(AREA, NOME_Municipio), uma vez que existam somente 16 possíveis valores de AREA. Essa representação reduziria à redundância de se repetir a mesma informação em milhares de tuplas de LOTES1A. A FNBC é uma forma normal mais restrita que não admitiria LOTES1A e sugeriria a decomposição da mesma, conforme ilustra a figura 9.11a.

(a) Normalização para FNBC com perda da df2 nas relações decompostas.

LOTES 1A




LOTES1AX

| #ID_Propriedade | Área | #Lote |
|---|---|---|

LOTES1AY

| Área | Nome_Municipio |
|---|---|

b) Relação R está na 3FN, mas não na FNBC.

R

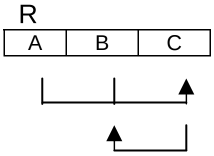



Figura 9.11(a, b) Forma normal de Boyce-Codd.

Considere agora as seguintes dependências funcionais da relação leciona da figura 12:

DF1: (aluno, disciplina) → instrutor
DF2: instrutor → disciplina

Note que (aluno, disciplina) é uma chave candidata para essa relação e que as dependências mostradas seguem o padrão da figura 9.11b. Portanto essa relação está na 3FN, mas não está na FNBC. A decomposição desse esquema em dois esquemas não é simples porque pode ser decomposto em um de três pares possíveis:

1. (aluno, instrutor) e (aluno, disciplina)
2. (disciplina, instrutor) e (disciplina, aluno)
3. (instrutor, disciplina) e (instrutor, aluno)

LECIONA

| Aluno | Disciplina | Instrutor |
|---|---|---|
| Narayan | Banco de Dados | Mark |
| Smith | Banco de Dados | Navathe |
| Smith | Sistemas Operacionais | Ammar |
| Smith | Teoria | Schulman |
| Wallace | Sistemas Operacionais | Ahamad |
| Wallace | Banco de Dados | Mark |
| Wong | Banco de Dados | Omiecinski |
| Zelaya | Banco de Dados | Navathe |

Figura 9.12 relação leciona está na 3FN, mas não na FNBC.

Todas as três decomposições “perdem” a dependência DF1. A mais desejável dessas três decomposições é a terceira, porque não irá gerar tuplas inválidas após uma junção. Em geral uma relação que não esteja na FNBC, deve ser decomposta de modo a satisfazer essa propriedade (de ser sem perda de junção), mesmo que isso signifique abster-se da preservação de todas as dependências funcionais. Esse tópico será tratado na seção 9.5.

Exercícios

1- Considere uma relação universal R = {A, B, C, D, E, F, G, H, I, J} e o conjunto de dependências funcionais F = {{A, B} → {C}, {A} → {D, E}, {B} → {F}, {F} → {G, H}, {D} → {I, J}}. Qual é a chave para R? Decomponha R em relações na 2FN e em seguida na 3FN.

2- Repita o exercício anterior com o seguinte conjunto de dependências funcionais G = {{A, B} → {C}, {B, D} → {E, F}, {A, D} → {G, H}, {A} → {I}, {H} → {J}}.

3- Prove que qualquer esquema de uma relação com dois atributos está na FNBC.

4- Considere a seguinte relação:

| A | B | C | TUPLA# |
|---|---|---|---|
| 10 | b1 | c1 | #1 |
| 10 | b2 | c2 | #2 |
| 11 | b4 | c1 | #3 |
| 12 | b3 | c4 | #4 |
| 13 | b1 | c1 | #5 |
| 14 | b3 | c4 | #6 |

a) Dada extensão da relação apresentada, qual(is) dependência(s) abaixo descritas podem ser mantida (s)? Se a dependência não pode se manter, explique o porquê, especificando as tuplas que causam violação.
A→ B
B→C
C →B
B→A
C→A

b) A relação mostrada anteriormente tem uma chave candidata? Justifique sua reposta.

5) Considere a relação R(A, B, C, D, E) com as seguintes dependências:
AB → C, CD → E, DE → B
Existem chaves candidatas para essa relação? Justifique sua resposta.

6) Considere a relação para livros publicados:

LIVRO (título_do_livro, nome_do_autor, tipo_do_livro, preço_de_tabela, afiliação_do_autor, editora)

Suponha as que existam as seguintes dependências:
título_do_livro → editora, tipo_do_livro
tipo_do_livro → preço_de_tabela
nome_do_autor → afiliação_do_autor

a ) Em que forma normal está a relação? Justifique sua resposta.
b) Aplique a normalização até que não possa mais decompor as relações. Justifique as razões de cada decomposição.

# 10 Data Warehouse – Uma visão geral

*Warehousing*[6] é uma técnica utilizada para recuperação e integração de dados a partir de fontes distribuídas, autônomas e, possivelmente, heterogêneas.

Estes dados são armazenados em um grande depósito chamado de *Data Warehouse*. Um data warehouse sumaria os dados que são organizados em dimensões, disponibilizando-os para consultas e análises através de aplicações OLAP (*On-Line Analytical Processing*) e sistemas de suporte à decisão.

Os data warehouses vêm sendo muito utilizados pelas empresas, já que proporcionam um alicerce sólido de integração de dados corporativos e históricos para a realização de análises gerenciais. Sua construção e implementação são feitas de uma maneira passo a passo, organizando e armazenado os dados sob uma perspectiva de longo prazo. Assim, partindo-se de dados históricos básicos, pode-se realizar análises de tendências.

Por sua característica básica – integração de dados provenientes de várias fontes diferentes – a etapa mais complexa na implementação de um data warehouse é o processo de carga. Neste processo, os dados distribuídos pelos vários ambientes operacionais (bases de dados de produção, que contêm os dados utilizados pelos vários sistemas transacionais de uma empresa) devem ser selecionados, trabalhados com o objetivo de padronização e limpeza, transferidos para o novo ambiente e finalmente carregados, sempre atendendo ao padrão da modelagem utilizada para o data warehouse. Este processo é feito periodicamente, sendo que sua freqüência depende de vários fatores relacionados ao modelo de negócios utilizado pela empresa e, normalmente, não é menor que 24 horas. Desta forma, podemos dizer que os dados armazenados no data warehouse são, para todos os propósitos práticos, uma longa série de visões do banco de dados, tiradas ao longo do tempo. Uma vez que os dados são armazenados no data warehouse, eles não mais sofrem atualizações, sendo, portanto, um ambiente apenas de carga e acesso.

Após sua criação e primeira carga, o data warehouse passa a sofrer cargas incrementais que devem refletir o ambiente operacional ao longo do tempo tornando-o um imenso repositório para os sistemas de apoio à decisão.

## 10.1 O que é o Data Warehouse

Um data warehouse existe para responder as questões que as pessoas têm sobre os negócios. Esta função contrasta fortemente com o propósito dos sistemas transacionais que as empresas utilizam e requer que o desenho ou o modelo de dados do data warehouse siga princípios completamente diferentes. As técnicas de modelagem dimensional, se aplicadas corretamente, garantem que o projeto do data warehouse reflita a forma de pensar dos gerentes de negócio e possa ser utilizado para atendê-los.

Em todas as empresas, o processo de criação de negócios é composto por uma série de eventos que caracterizam suas atividades principais. A natureza e freqüência destes eventos variam conforme o tipo de negócio em que uma empresa está envolvida: um produto é manufaturado, uma conta é creditada enquanto outra é debitada, um assento é reservado, um pedido é incluído etc. O controle e processamento corretos destes eventos são críticos para uma empresa, sendo que estas atividades contribuem para seu sucesso ou fracasso. Para isso, a maioria das organizações possui um conjunto de sistemas conhecidos como *sistemas transacionais* ou *sistemas OLTP* (*OnLine Transaction Processing*) que capturam os eventos de forma individual e todos os detalhes associados a eles. Cada um destes sistemas está encarregado de um tipo diferente de atividades e trata das transações de negócios segundo um conjunto de regras que garanta sua consistência e o armazenamento de todos os detalhes associados. Para um sistema OLTP, os princípios de desenho como normalização e consistência de transações são extremamente críticos. Porém, por tratar as transações de forma individual, os sistemas OLTP falham no que se refere a questões sobre o processo do

[6] O termo *Warehousing* não possui tradução adequada para o português.

negócio, como “quais foram os produtos mais vendidos durante o mês passado?” ou “quais produtos estão perdendo *market share?*” ou ainda “quais são nossos clientes mais fiéis?”.

O data warehouse é construído para responder questões que não estão limitadas às transações individuais, porém, tratam do processo como um todo. Para isso, o desenho do data warehouse deve refletir a forma com que os especialistas enxergam o negócio e este é o ponto chave que distingue um data warehouse de um sistema OLTP. A técnica utilizada para se obter um modelo para o data warehouse que identifique e represente a informação importante para o modelo de negócios é a *modelagem dimensional*. Quando bem definido, o modelo dimensional pode ser uma ajuda de valor incalculável para as áreas de negócio, apoiando e otimizando todo o processo de tomada de decisões.

Um data warehouse corporativo pode ser definido em termos de seis características básicas que o diferenciam dos outros sistemas na empresa. Os dados em um data warehouse são:

1. *Separados* dos sistemas transacionais da empresa e populados a partir destes;
2. *Disponíveis*, na sua totalidade, para a atividade de serem interrogados pelos usuários de negócios;
3. *Integrados* para ser uma base única e padrão para o modelo da empresa;
4. *Associados à informação temporal* e a períodos de tempo definidos, como fechamentos mensais ou baseados no ano fiscal;
5. *Orientados por assunto*, ou seja, organizado para descrever o desempenho do negócio;
6. *Acessível* aos usuários que tenham um conhecimento limitado de sistemas computacionais ou estruturas de dados.

Além destas características, podemos citar sua *não-volatilidade*, ou seja, uma vez que o dado foi carregado no data warehouse, não deve mais sofrer alterações.

## 10.2 O modelo dimensional e suas implementações

A modelagem de um data warehouse normalmente utiliza uma abordagem diferente da modelagem entidade-relacionamento, que é a maneira convencional para desenhar uma base de dados relacional contemplando toda a teoria de normalização dos dados.

O modelo dimensional (também chamado de multidimensional), utilizado para definir um data warehouse, representa os indicadores importantes para uma área de negócios, que são chamados de fatos ou métricas e os parâmetros, chamados dimensões, através dos quais estas métricas são vistas. A Figura 1 mostra um exemplo de modelo dimensional para um processo de pedidos. As métricas definidas estão no quadro central e as dimensões estão representadas nos quadros ao redor das métricas. As métricas são sumariadas ou detalhadas de acordo com o interesse da análise a ser feita sobre os dados. Este modelo é fácil de ser entendido por uma pessoa da área de negócios, já que “as coisas que eu avalio” estão na parte central do diagrama e “as formas de se olhar para elas” estão nos quadros em volta.

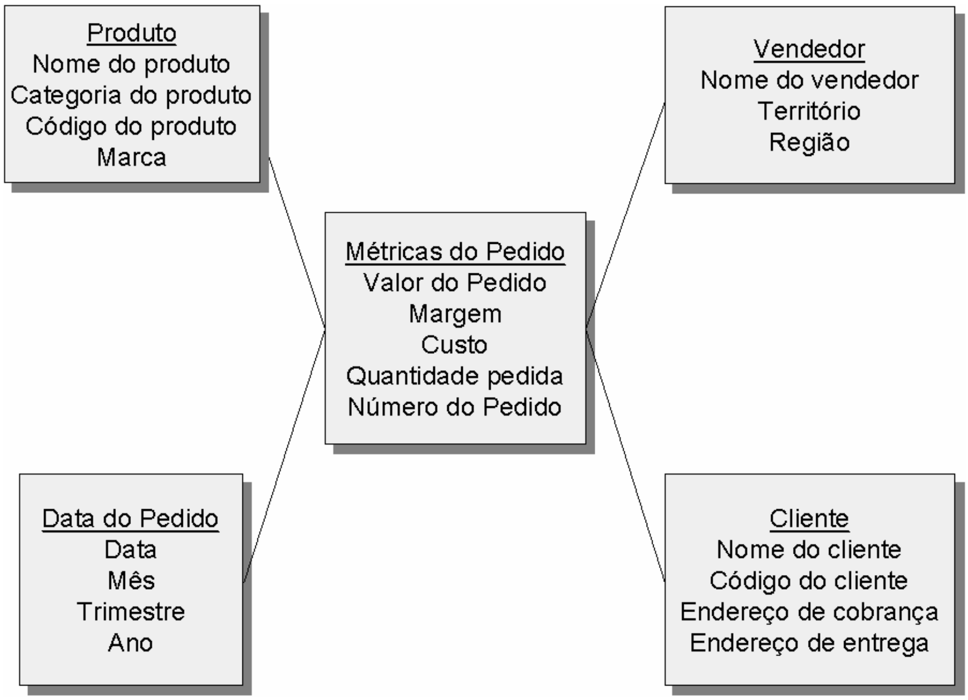



***Figura 1 - Modelo dimensional para um processo de pedidos***

È de fácil percepção que estes quadros facilmente se transformarão em tabelas, que podem ser utilizadas para armazenar toda a informação. Um modelo como este não muda muito ao ser implementado em um banco de dados relacional. Cada quadro com os atributos de uma dimensão se torna uma tabela, chamada de *tabela dimensão,* na base de dados e o quadro central se torna uma grande tabela, chamada *tabela fato*, que contém, por vezes, milhões ou bilhões de linhas.

Contudo, os modelos dimensionais nem sempre são implementados em bases de dados relacionais. Existem no mercado os *bancos de dados multidimensionais*, ou *MDDBs*, que armazenam as informações em um formato diferente, freqüentemente chamados de *cubos*. Os cubos são construídos de tal forma que, cada combinação de atributos das dimensões com uma métrica ou é pré-calculado ou é calculado muito rapidamente. Entretanto, a natureza de uma base de dados multidimensional também significa que não é possível manipular volumes de dados extremamente grandes já que, uma transação de análise dos dados, com uma ferramenta OLAP, que envolva um grande volume de dados vai consumir grande quantidade de memória ou simplesmente não se efetua. Além disso, o número de atributos dimensionais armazenados em um cubo pode impactar o tamanho e o desempenho do cubo.

Uma das alternativas para solucionar estes problemas pode ser a implementação do modelo dimensional em um banco de dados relacional e, após isto, utilizá-lo como fonte para os cubos. Esta abordagem é muito utilizada em empresas que querem executar análises em pequenos subconjuntos de um grande conjunto de dados armazenados em um data warehouse. Quando esta abordagem é implementada, o data warehouse como um todo fica armazenado no banco de dados relacional, enquanto que os cubos, contêm partes ou segmentos do data warehouse que são chamados de *data marts*.

Uma outra alternativa é utilizar uma ferramenta de consulta acessando o data warehouse no banco relacional transformando o resultado da consulta em um cubo que permita a análise rápida dos dados. Estes cubos podem ser gerados no computador do usuário ou em um servidor de aplicações. Esta abordagem é interessante para os usuários, já que não requer a administração centralizada destes cubos. É importante notar, porém, que o usuário fica sujeito aos limites de capacidade de processamento de seu PC ou do servidor de aplicações.

Uma arquitetura para o Data Warehouse é apresentada na Figura 2. Nesta arquitetura, os dados são provenientes dos sistemas operacionais. Estas fontes são conectadas a *wrappers* ou monitores que efetuam o processo de seleção, transformação e limpeza dos dados. Além disso, monitoram as alterações nas fontes de dados propagando-as a um componente integrador, que combina os dados selecionados a partir das diferentes fontes operacionais. Estes dados, já consistentes, são propagados para o warehouse.

O metadados contém informações relevantes sobre a criação, gerenciamento e uso do data warehouse e funciona como uma ponte entre os usuários do data warehouse e os dados nele contidos.

O warehouse pode ser acessado através de um servidor OLAP, que tem como objetivo, apresentar as informações multidimensionais para as ferramentas de acesso, análise, geradores de relatórios, planilhas e ferramentas de mineração de dados (*Data Mining tools).* Basicamente, o servidor OLAP interpreta as consultas dos usuários convertendo-as em instruções adequadas, muitas vezes complexas, para o acesso ao data warehouse. Atualmente existem dois tipos de solução para implementação para acesso ao repositório de data warehouse. Uma é a utilização de modelos relacionais associados a tecnologias de buscas multidimensionais em cubos pré-construídos (MOLAP). Outra, é a utilização de gerenciadores relacionais incrementados com tecnologias de índices bitmap e recuperação de dados com listas invertidas (ROLAP).

Na Figura 2 é possível também identificar as várias camadas que compõem tal arquitetura:

1. Ferramentas de acesso para os usuários;
2. Servidor OLAP (MOLAP-ROLAP)
3. Sistema de gerenciamento do data warehouse e ferramentas para selecionar, transformar, limpar, integrar e copiar os dados;
4. Dados dos sistemas operacionais.

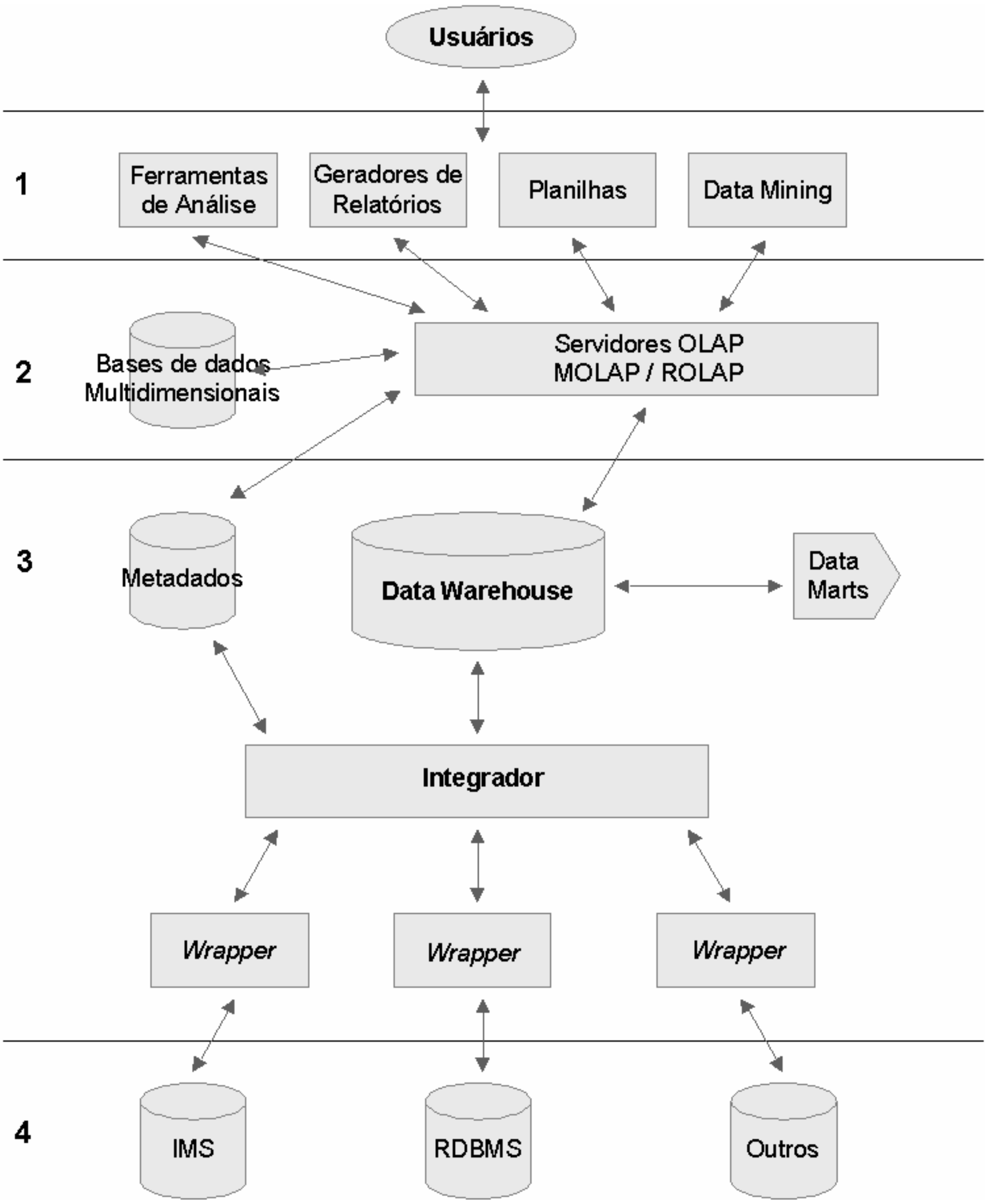



***Figura 2 – Arquitetura de data warehouse proposta por* [17]**

## 10.2.1 O modelo formal da base de dados multidimensional

O modelo multidimensional implementado em uma base de dados relacional foi formalmente definido por (Baralis, Paraboschi &Teniente), e será apresentado nesta seção.

A estrutura básica deste modelo pode ser representada por um diagrama entidade-relacionamento como na Figura 3.

**Definição 2.3.1** *Uma* **base de dados multidimensional** *é uma coleção de relações $D_1$,..., $D_n$, F, onde:*

- *Cada $D_i$ é uma* **tabela dimensão***, isto é, uma relação caracterizada por um identificador que identifica unicamente cada tupla ($d_i$ é a chave primária de $D_i$).*
- *F é uma* **tabela fato***, isto é, uma relação que conecta todas as tabelas $D_1$,..., $D_n$; o identificador de F é composto pelas chaves estrangeiras $d_i$,..., $d_n$ de todas as tabelas dimensão conectadas. O esquema de F contém um conjunto de atributos*

*adicionais V (que representam os valores sobre os quais serão aplicadas as funções de agregação).*

As tabelas dimensão podem conter hierarquias.

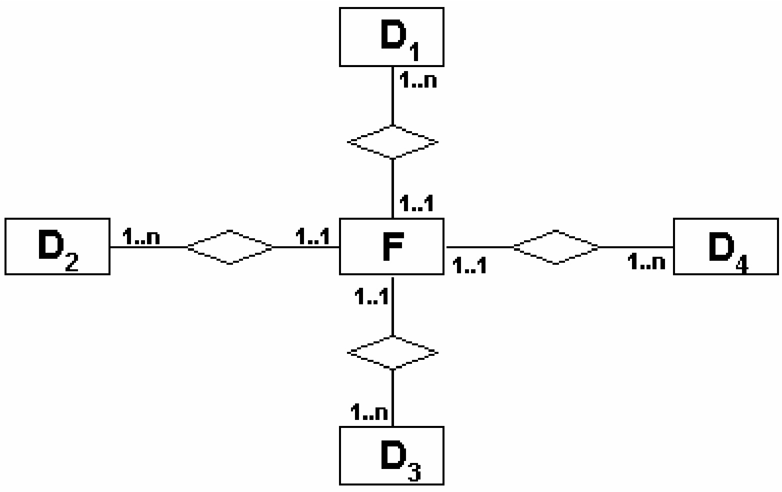



***Figura 3 – Representação Entidade-Relacionamento de uma base de dados multidimensional***

**Definição 2.3.2** *Seja D uma tabela dimensão com o identificador d. Uma* **hierarquia de atributos** *em D é um conjunto de dependências funcionais* $FD_D = \{fd_0, fd_1, ..., fd_n\}$*, onde cada* $fd_i$ *é caracterizado por dois conjuntos de atributos* $A^l_i \subset$ Attr($D$) *e* $A^r_i \subset$ Attr($D$) *(chamados respectivamente de lado esquerdo e lado direito da dependência); a dependência é representada por* $fd_i : A^l_i \to A^r_i$.

Cada dependência funcional $fd_i$ é uma restrição ao conteúdo da tabela dimensão $D$, sendo que: para cada par de tuplas $t_1$ e $t_2 \in D$, $t_1 [A^l_i] = t_2 [A^l_i] \Rightarrow t_1 [A^l_i] = t_2 [A^r_i]$ . Uma dependência $fd_0$ com $A^l_0 = \{d\}$ e $A^r_0 = \{Attr(D) - d\}$ estará sempre presente em $FD_D$. As dependências funcionais devem ser acíclicas, isto é, o grafo obtido pelo desenho de um arco partindo de $a_x$ e chegando em $a_y$, deve ser acíclico, se $\exists fd_i \in FD_D \mid a_x \in A^l_i \wedge a_y \in A^r_i$.

> **Exemplo:** Vamos considerar um exemplo prático, a base de dados multidimensional para uma grande cadeia de lojas de varejo. Com um grande número de lojas, cada uma delas sendo um supermercado que vende uma ampla variedade de diferentes produtos. A base de dados multidimensional armazena informação sobre cada venda, por loja, por dia e considera também as promoções em cada produto vendido. Podemos identificar as seguintes dimensões: *Product*[7], que caracteriza cada produto vendido, *Store*, que caracteriza cada ponto de venda, *Time* e *Promotion*, que descreve as características das promoções dos produtos. A tabela fato fornece as informações sobre as vendas, sobre as quais serão realizadas as análises financeiras. Inclui identificadores para todas as dimensões e vários atributos descrevendo as vendas (como valor da venda, quantidade vendida, etc.)
>
> Se considerarmos a dimensão *Store*, do exemplo acima, com chave *s* e restrita a um conjunto de atributos $\{z, c, s_t, n\}$, que representam respectivamente *zip code*, *county*, *state* e *number of sale clerks*. A dimensão tem a seguinte hierarquia de atributos:
>
> $$\{fd_0 : s \to \{z, c, s_t, n\}, fd_1 : z \to c, fd_2 : c \to s_t\}$$

**Definição 2.3.3** *Uma* **hierarquia de atributos da base de dados multidimensional** $FD_{DB}$ *é a união das hierarquias de atributos* $DF_{Dj}$ *de todas as dimensões* $D_j$ *existentes na base de dados multidimensional.*

---

[7] Por razões didáticas, os nomes de tabelas e atributos serão mantidos no idioma original, inglês.

## 10.3 Aspectos da Modelagem Dimensional

### 10.3.1 Características

Conforme já comentado, o modelo dimensional é poderoso, pois reflete a maneira de pensar dos especialistas de negócios e responde às suas necessidades de informações. A tecnologia relacional de bancos de dados possibilita ao data warehouse ser utilizado para responder as questões de forma rápida e precisa. Para isso, são necessários três componentes essenciais, a saber:

1. Os dados provenientes das várias fontes distribuídas pela empresa e armazenados em um único local;
2. Ferramentas que possibilitem a análise das informações armazenadas de forma rápida, flexível com alta qualidade de apresentação e
3. O conhecimento do especialista de negócios.

Existem inúmeras ferramentas, como as citadas no item 2 acima, disponíveis no mercado e chamadas de OLAP. Estas ferramentas permitem ao usuário visualizar os vários níveis de detalhamento da informação, sob as visões das diferentes dimensões definidas no modelo e têm sido alvo de vários trabalhos acadêmicos. O conhecimento do especialista de negócios é outro componente essencial, já que apenas ele pode tirar as conclusões das informações apresentadas, com o objetivo de tomada de decisões da corporação.

Em linhas gerais, o processo de implementação de um data warehouse está dividido nas seguintes etapas:

1. Levantamento do processo de negócio a modelar;
2. Definição dos modelos conceitual, lógico e físico;
3. Definição do processo de carga;

Dentre todas as etapas citadas acima, a que envolve o processo de carga é de longe a mais complexa. Além de trazer os dados de vários sistemas transacionais diferentes, o processo engloba atividades de verificação, padronização, limpeza e transformação dos dados antes da carga. A definição da periodicidade da carga depende da natureza do negócio que compõe o data warehouse e do tipo de informação armazenada. Algumas áreas de negócios requerem carga diária, enquanto que outras necessitam apenas de cargas mensais. Seja qual for a periodicidade escolhida, fica claro que o data warehouse não está sincronizado em tempo real com os sistemas transacionais. Para algumas aplicações, esta característica estática do data warehouse não compromete o resultado das análises porém, para outras, um data warehouse dinâmico, ou seja, sincronizado com os sistemas transacionais, é essencial. Para implementar um data warehouse dinâmico seria necessário definir uma arquitetura que permitisse propagar as atualizações nas bases transacionais no instante em que elas ocorrem.

Atualmente, pode-se dizer que os data warehouses implantados são de natureza estática, tendo processos de carga de alta complexidade e que requerem um grande tempo de processamento.
As próximas seções detalham vários aspectos envolvidos na modelagem do data warehouse.

### 10.3.2 Conceitos da modelagem

A modelagem dimensional combina tabelas fato que armazenam dados históricos temporais (normalmente numéricos), indexadas por chaves dimensionais que estão descritas nas tabelas dimensão correspondentes. As tabelas dimensão contêm informações como, por exemplo: períodos de tempo, produtos, mercados, organizações, contas, vendedores e clientes e inclui as descrições e os atributos destas dimensões. Além disso, as tabelas dimensão contemplam toda a estrutura da dimensão, como os agrupamentos dos produtos em marcas e em categorias, as cidades em estados, em regiões e em países e assim por diante.

As tabelas fato contêm as métricas ou os fatos a serem analisados dentro do modelo de negócios. Cada um destes fatos está diretamente relacionado às dimensões, que descrevem suas condições de ocorrência. Todo o relacionamento entre a tabela fato e as tabelas dimensão é feito por meio de chaves.

Uma consulta executada neste modelo dimensional, geralmente se inicia por uma pesquisa nas tabelas dimensão, aplicando-se os filtros de valores e obtendo-se como resultado um conjunto de chaves. Após isso, o acesso é feito à tabela fato, garantindo assim a precisão no acesso aos dados através de uma estrutura completa de chaves, eliminando-se um *table scan* e, com isso, obtendo-se o melhor desempenho possível de uma tecnologia relacional. O conceito de armazenamento das dimensões separadamente garante que a base de dados trate os vetores esparsos de maneira eficaz, isto é, sem armazenar vazios, assegurando o acesso mais eficiente possível.

O modelo dimensional pode ser implementado utilizando vários tipos de esquemas diferentes. Provavelmente, o *star schema*, apresentado por R. Kimball, seja o primeiro esquema utilizado para representar o data warehouse implementado em um banco de dados relacional.

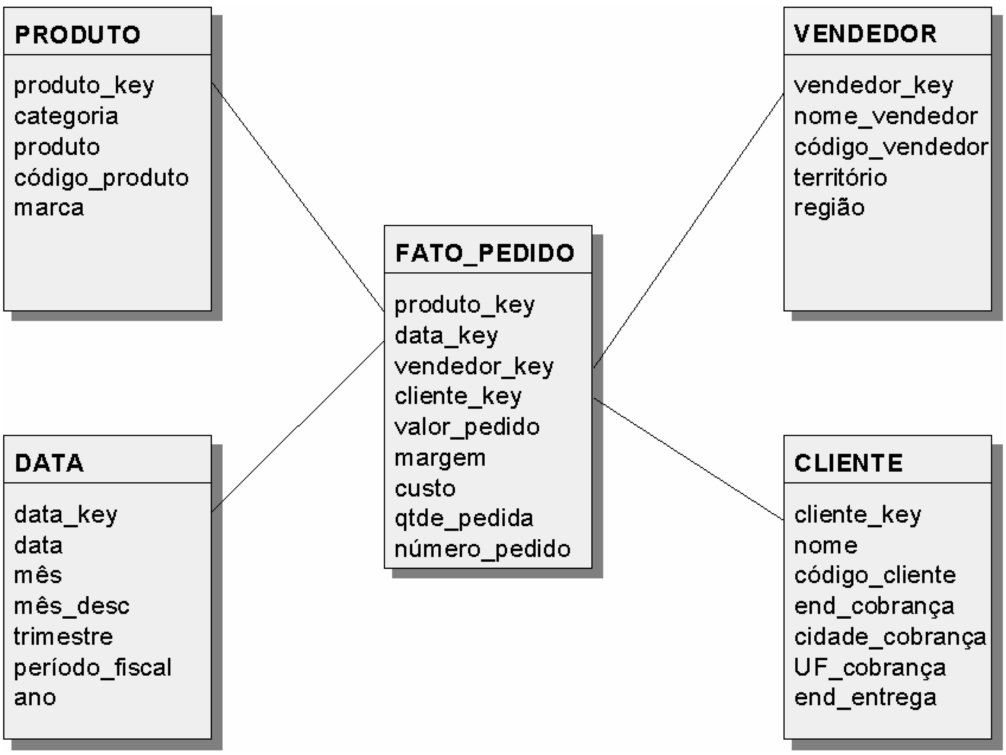



***Figura 4 - Esquema para um processo de pedidos***

Apesar de ser o mais conhecido, o star schema não é o único. De fato, existe uma série de variações que serão analisadas a posteriori. Porém, antes de iniciar a modelagem em um esquema particular, temos que definir o processo de negócio a ser modelado.

A Figura 4 mostra um exemplo de esquema para um processo de pedidos. A tabela central é a tabela fato e as tabelas ao redor desta são as dimensões. Neste exemplo estão representadas as dimensões Produto, Tempo (data), Cliente e Vendedor (incluindo informações de ordem geográfica). Os fatos representados são: valor do pedido, margem, custo, quantidade pedida e número do pedido.
Depois de definido o escopo do data warehouse, os próximos passos são: definir a granularidade das informações representadas na tabela fato, identificar as dimensões e enumerar as métricas ou fatos. A seguir daremos uma descrição sucinta sobre cada uma destas etapas e, na seqüência, mostraremos os tipos de esquemas básicos e suas variações.

## 10.3.2.1 A granularidade das informações

Como já dito, uma das primeiras decisões feita para modelar o data warehouse está relacionada com o nível de detalhe das métricas a serem armazenadas. Este nível de detalhamento é conhecido como granularidade da tabela fato. É muito importante que todas as

linhas na tabela fato sejam armazenadas com informações exatamente no mesmo nível de detalhes. Como exemplo, um processo de pedidos teria sua granularidade definida no nível de detalhe da linha individual do pedido. Se forem armazenados diferentes níveis de detalhamento na tabela fato, a utilização da base de informações ficaria bastante prejudicada. Suponha que a maioria das informações no nível da linha de detalhe do pedido esteja armazenada, sendo que algumas informações foram armazenadas no nível do cabeçalho do pedido. No nível de detalhe da linha do pedido existe um relacionamento com cliente, produto, representante comercial e tempo. Porém, no nível do cabeçalho do pedido, o relacionamento com os produtos específicos por pedido desaparece. As informações relevantes que estejam em diferentes níveis de granularidade deveriam ser armazenadas em tabelas fato diferentes.

Em geral, podemos dizer que, a aderência de uma tabela fato a uma certa granularidade requer que as chaves que relacionam as linhas da tabela fato às linhas das dimensões nunca sejam nulas. Um relacionamento opcional a uma dimensão geralmente indica um problema de granularidade.

#### 10.3.2.2 As dimensões

Uma vez que o nível de granularidade esteja definido, a escolha das dimensões a serem utilizadas deve ser o próximo passo. O ideal seria definir uma dimensão com um grande número de atributos, representando um rico conjunto de detalhes sobres o processo de negócio (é comum que as dimensões apresentem 100 ou até mesmo 200 atributos em uma única tabela dimensão).

As tabelas dimensão contêm informações sobre as dimensões dos dados, ou seja, tempo, produto, mercado, contas e assim por diante e devem ser desenhadas a partir da perspectiva do usuário. Por esta razão, os atributos e suas descrições devem ser definidos de forma significativa para o usuário e adequados para a posterior exibição em relatórios. As principais funções da tabela dimensão são reunir os atributos que serão utilizados para qualificar as consultas e cujos valores serão utilizados para agrupar e sumariar as métricas.

Cada tabela dimensão deve ter múltiplos atributos que contenham valores ou textos que possam ajudar a descrever a chave. Estas colunas de atributos são utilizadas para filtrar o conteúdo da dimensão. Além disso, a utilização de atributos do tipo inteiro, quando apropriados, favorecem as operações de filtragem como *maior que*, *menor que* e *entre*.

Os atributos de uma dimensão podem compor uma hierarquia ou ser apenas descritivos. Por exemplo, em uma dimensão *produto*, a hierarquia pode ser composta pelos atributos *item*, *marca*, *tipo* e *divisão*. A hierarquia definida na dimensão é requisito básico para as funções de agregação das métricas contidas na tabela fato. Através dos agrupamentos dos elementos na hierarquia, os usuários podem analisar os fatos em um nível maior ou menor de detalhes, conforme sua necessidade específica. Este conceito de hierarquia, que permite a implementação de agregações das métricas, é muito importante também para a dimensão *Tempo* e, vale ressaltar que é muito raro um modelo dimensional que não inclua a dimensão *Tempo* como uma dimensão fundamental.

Deve-se resistir ao impulso de aplicar as regras de normalização nas tabelas dimensão. O processo de normalização, separando os atributos em várias tabelas diferentes faz com que as consultas fiquem bem mais complexas, o banco de dados gaste mais tempo para recuperar os dados e, por conseqüência, os usuários esperem mais por suas respostas. Este custo é muito alto como resultado da economia de apenas uns poucos bytes em uma tabela dimensão que, em comparação com uma tabela fato, é minúscula.

Manter as tabelas dimensão desnormalizadas faz com que as hierarquias naturais contidas nos dados fiquem bem definidas. No exemplo anterior, de pedidos, nota-se que alguns exemplos como a dimensão Tempo (Data) inclui os atributos Ano, Trimestre, Mês e Data (que inclui o dia), juntamente com outros atributos relacionados a uma data específica. Estes componentes não foram colocados em uma série de tabelas progressivamente mais detalhadas. Em vez disso, estão em uma única tabela que deixa clara a hierarquia. Por exemplo, enquanto que a ocorrência do ano de 2004 possa aparecer na tabela 365 vezes, isto ficará invisível ao usuário. Quando as métricas forem sumariadas por ano, apenas uma ocorrência de 2004 aparecerá no relatório.

Um atributo muito importante da tabela dimensão é sua chave. A chave primária de uma tabela dimensão dever ser sempre um atributo único e definido pelo sistema com um valor genérico. Por questões de desempenho, não se utilizam chaves compostas por várias partes nem

tampouco chaves concatenadas. Também não são utilizados as chaves ou os identificadores provenientes de outros sistemas como, por exemplo, código do cliente ou código do produto. Existem várias razões para se utilizar chaves genéricas em vez de chaves com valores significativos. Em caso de alterações de atributos de um cliente, por exemplo, seu endereço, um tratamento especial seria necessário e a inserção deste mesmo cliente, com seu novo endereço, com outra chave. Se o código do cliente for utilizado como chave primária isto não será possível. Os tratamentos de alterações serão analisados neste trabalho, em seções posteriores.

#### 10.3.2.3 Os fatos

Após identificação da granularidade e das dimensões, é o momento de focalizar a tabela fato. Para isso, começa-se definindo quais métricas ou fatos deseja-se avaliar no data warehouse. Estes fatos são os números que serão analisados através das diferentes dimensões. A seleção dos fatos para compor o modelo do data warehouse é relativamente simples: uma vez que a área de negócios esteja definida, a lista de fatos a serem utilizados responde a questão: "*O que estamos avaliando?"*.

O exemplo, representado pelas Figuras 1 e 4, ilustra em suas respectivas tabelas centrais, quais são os valores relevantes para a análise do processo de pedidos.

Existem três tipos de métricas ou fatos, discutidos na próxima seção.

### 10.3.3 Os três tipos de métricas

As métricas mais comuns são as *completamente aditivas.* Diz-se que uma métrica é completamente aditiva, quando faz sentido sumarizá-la adicionando seus valores ao longo de qualquer dimensão. No exemplo de pedidos, o valor do pedido, a margem, o custo e a quantidade pedida são todas métricas completamente aditivas. Apesar de serem armazenadas na tabela fato para um determinado produto, um cliente, um vendedor e uma data específica, pode-se facilmente sumarizá-las da maneira que for de interesse. Para verificar os pedidos de um determinado mês, basta adicionar os valores dos pedidos de todas as datas daquele mês. O mesmo ocorre para verificar a margem obtida para uma determinada categoria de produtos. Basta adicionar os valores de margem para todos os produtos de uma determinada categoria.

As métricas completamente aditivas são bastante úteis e poderosas, já que não existem limitações em sua utilização. Podem ser facilmente sumariadas para qualquer combinação de valores das dimensões.

Em contraste total com este tipo de métrica, têm-se as métricas *não aditivas*, que não podem ser adicionadas ao longo dos valores das dimensões. Considere a métrica margem, expressa como uma porcentagem de vendas, também denominada *margem percentual*. Esta métrica representa a margem como percentual e não mais como valor expresso na moeda corrente. Seria muito simples adicionar esta métrica à tabela fato, porém, esta teria pouca utilidade, já que não poderia sumarizá-la de acordo com a dimensão de interesse. Por exemplo, em uma determinada data, um vendedor vende a um cliente 4 tipos diferentes de produtos, cada um deles com uma margem percentual de 25%. Não faz sentido incluir os quatro valores de margem percentual para calcular a margem total para este pedido.

Aparentemente, conclui-se que este tipo de métrica não pode ser utilizada na tabela fato do exemplo acima. Na verdade, esta métrica é derivada de outras duas métricas que são aditivas: margem e valor do pedido. A solução é, portanto, armazenar na tabela fato apenas seus componentes, que são completamente aditivos, sendo que o cálculo para expressar a margem percentual deverá ser feito pela aplicação, que neste caso é a divisão da margem pelo valor.

O terceiro tipo de métrica é a *semi-aditiva*. A métrica semi-aditiva pode ser sumariada ao longo de determinadas dimensões, porem não todas. Considere como exemplo o gerenciamento de saldo das contas de um banco. O saldo é armazenado no final de cada dia, para cada cliente, por conta ao longo do tempo. Em alguns casos este saldo é aditivo. Se um cliente tem uma conta corrente e uma conta poupança, pode-se adicionar os saldos de cada conta no final de um dia e obter resultado significativo. É possível também adicionar os saldos de uma determinada agência para obter um panorama da situação geral de cada localidade. Entretanto, não faz o menor sentido adicionar o saldo de um cliente ao longo do tempo. Por esta razão, a métrica saldo é considerada semi-aditiva.

Esse tipo de atributo não aditivo ou semi-aditivo pode ser agregado ou sumariado utilizando-se outros operadores como média, máximo ou mínimo. É o caso da métrica temperatura, que é considerada não aditiva, já que adicionar temperaturas dificilmente faz sentido.

### 10.3.4 Outros elementos da tabela fato

Além das métricas, cada tabela fato tem uma chave primária. Esta chave primária é composta por várias colunas, sendo que cada uma delas corresponde logicamente a uma chave na tabela dimensão. Cada elemento componente da chave deve, portanto, estar representado e descrito em uma tabela dimensão correspondente, que logicamente se une a uma ou mais tabelas fato através de colunas idênticas. Então, a chave primária de uma tabela fato é uma chave composta por um subconjunto de chaves estrangeiras para as tabelas dimensão. A dimensão *Tempo* é sempre representada como parte da chave primária.

A Figura 5 ilustra uma tabela fato que contém o valor das vendas e a quantidade de unidades vendidas, logicamente unida a uma tabela dimensão de *Mercado* correspondente. Esta Figura não representa as dimensões *Produto* e *Tempo*.

**Tabela Fato**

| TEMPO KEY | PRODUTO KEY | MERCADO KEY | Valor | Quantidade |
|---|---|---|---|---|
| 980715 | 101 | 4030 | 348,00 | 140 |
| 980716 | 101 | 4030 | 287,00 | 114 |
| 980716 | 101 | 4010 | 443,00 | 170 |
| 980717 | 101 | 2010 | 580,00 | 232 |
| 980718 | 101 | 2010 | 686,00 | 274 |

**Dimensão Mercado**

| MERCADO KEY | Mercado Descrição | Região | Estado | Cidade | Loja | Nível |
|---|---|---|---|---|---|---|
| 1010 | Região Sul | 1 | | | | 4 |
| 1020 | Região Sudeste | 2 | | | | 4 |
| 2010 | SP | 2 | 10 | | | 3 |
| 2020 | RJ | 2 | 20 | | | 3 |
| 3010 | São Paulo | 2 | 10 | 101 | | 2 |
| 3020 | Campinas | 2 | 10 | 102 | | 2 |
| 4010 | Loja ABC | 2 | 10 | 101 | 1001 | 1 |
| 4020 | Loja DEF | 2 | 10 | 102 | 1002 | 1 |
| 4030 | Loja GHI | 2 | 10 | 102 | 1003 | 1 |

***Figura 5 – Junção lógica entre tabelas fato e dimensão***

Ocasionalmente, não é necessário armazenar nenhuma métrica em uma tabela fato. Por vezes, o simples armazenamento de um relacionamento entre as dimensões em um certo ponto do tempo é tudo que uma área de negócios necessita como métrica. Este tipo de tabela fato é a tabela sem fato (do original *factless fact table*). O exemplo mais típico deste tipo de tabela fato é o controle de freqüência de alunos em determinadas disciplinas. Não existe uma métrica associada. Apenas a existência do relacionamento entre as dimensões já é um indicativo suficiente para o processo de análise. Este tipo de tabela encontra-se ilustrado na Figura 6.

Pode ocorrer que a mesma tabela fato tenha múltiplas chaves estrangeiras, como parte de sua chave primária, que se relacionam com a mesma dimensão. Por exemplo, o mesmo fato ter múltiplas datas associadas a ele, como no caso da tabela fato de remessa de produtos, onde

duas chaves estrangeiras, *data da remessa* e *data do pedido,* estão associadas a uma mesma tabela dimensão *tempo.* A Figura 7 ilustra esta situação.

Neste caso, não é necessária a criação de duas tabelas dimensão *data*. Em vez disso, visões ou sinônimos podem ser utilizadas para referenciar duas cópias virtuais da mesma tabela dimensão em uma única instrução SQL, no momento da consulta.

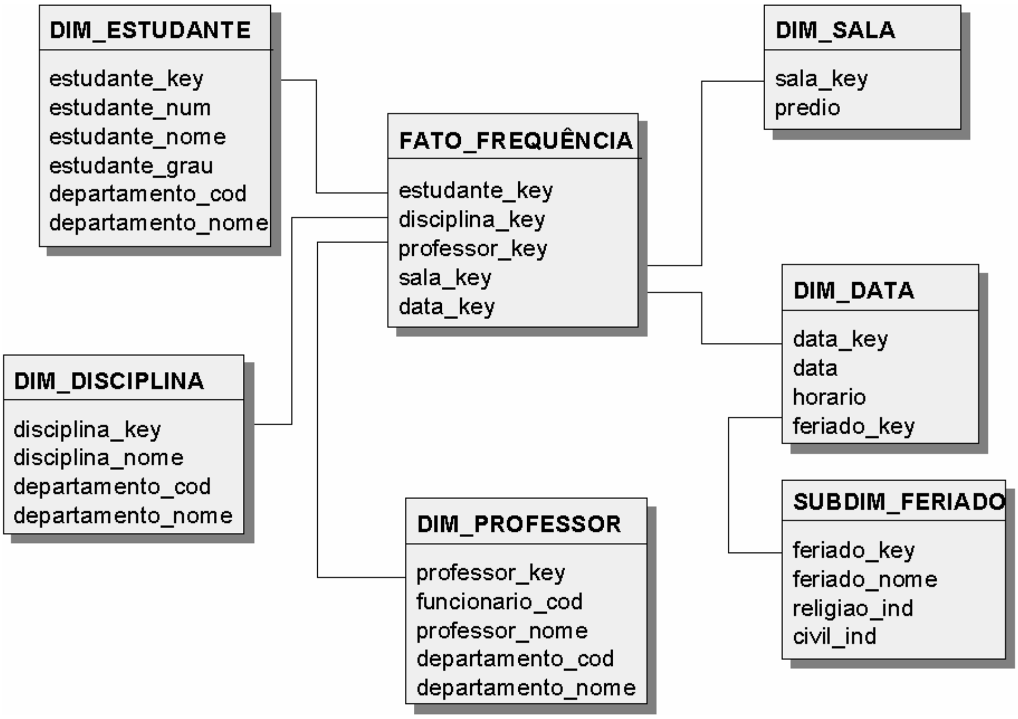



***Figura 6 – Um exemplo de tabela fato do tipo factless***

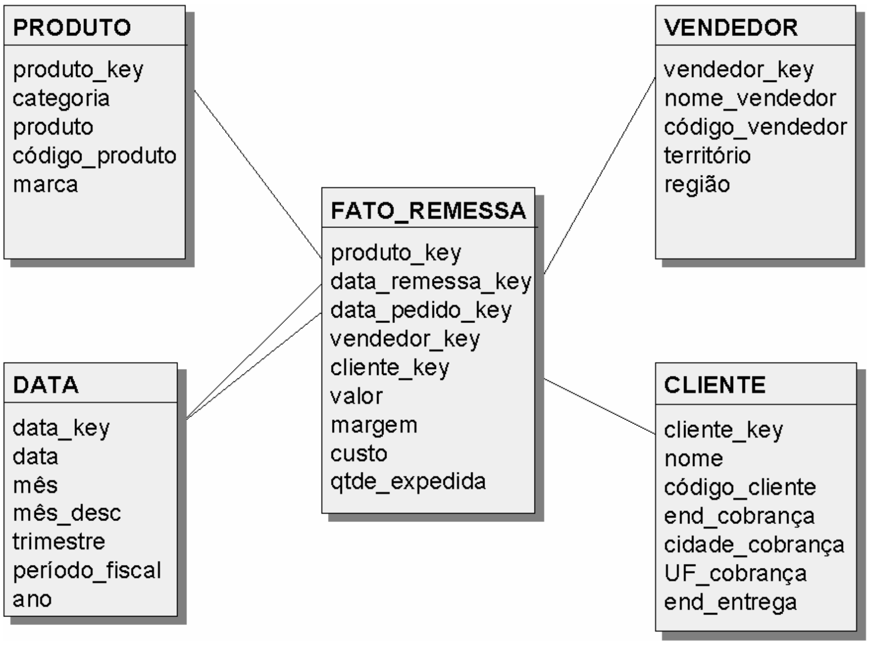



***Figura 7 –Relacionamento múltiplo entre uma tabela fato e uma dimensão***

## 10.4 Os esquemas básicos e suas variações

Após a definição do modelo conceitual, inicia-se o modelo lógico e sua conseqüente implementação física. A implementação física do modelo dimensional, em uma base de dados relacional, pode ser feita de várias formas diferentes. No entanto, para todos os esquemas, os fatos e as dimensões são implementados em tabelas físicas, onde cada linha é única e identificada por uma chave genérica, já discutida anteriormente. Nas próximas seções, os esquemas básicos, conhecidos como *star schema* e *snowflake schema*, e suas variações serão analisados.

### 10.4.1 O esquema Star clássico

Provavelmente, o *Star schema*, apresentado por R. Kimball, seja o primeiro esquema utilizado para projetar um data warehouse implementado em um banco de dados relacional. Neste esquema, a tabela fato contempla as seguintes características:

- Uma chave primária composta, sendo um elemento da chave para cada dimensão;
- Cada elemento chave para a dimensão deve ser representado – e descrito na tabela dimensão correspondente, para efetuar o *join*;
- Opcionalmente, uma ou mais métricas ou fatos a serem avaliados;
- Pode armazenar as métricas sumariadas em diferentes níveis de agregação.

Uma tabela dimensão é definida para cada dimensão do negócio a ser analisada, contendo:

- Uma chave genérica, gerada pelo sistema;
- Uma coluna de descrição genérica para a dimensão ou colunas de descrição para cada atributo da hierarquia;
- Atributos que permitam efetuar os filtros;

- Um indicador *nível*, para estabelecer o nível da hierarquia a que se refere a linha da tabela;
- Valores nulos ocasionais, dependendo do nível hierárquico da linha.

O objetivo de se armazenar toda a hierarquia é possibilitar, durante as consultas, a utilização de filtros que permitam a agregação ou sumário dos fatos nos vários níveis hierárquicos.

a.

| GEOGRAFIA_KEY | Geografia Descrição | Região | Estado | Cidade | Loja | Nível |
|---|---|---|---|---|---|---|
| 1010 | Região Sul | 1 | | | | 4 |
| 1020 | Região Sudeste | 2 | | | | 4 |
| 2010 | SP | 2 | 10 | | | 3 |
| 2020 | RJ | 2 | 20 | | | 3 |
| 3010 | São Paulo | 2 | 10 | 101 | | 2 |
| 3020 | Campinas | 2 | 10 | 102 | | 2 |
| 4010 | Loja ABC | 2 | 10 | 101 | 1001 | 1 |
| 4020 | Loja DEF | 2 | 10 | 102 | 1002 | 1 |
| 4030 | Loja GHI | 2 | 10 | 102 | 1003 | 1 |

b.

| GEOGRAFIA_KEY | Região | Região_desc | Estado | Estado_desc | Cidade | Cidade_desc | Loja | ... | Nível |
|---|---|---|---|---|---|---|---|---|---|
| 1010 | 1 | Região Sul | | | | | | ... | 4 |
| 1020 | 2 | Região Sudeste | | | | | | ... | 4 |
| 2010 | 2 | Região Sudeste | 10 | São Paulo | | | | ... | 3 |
| 2020 | 2 | Região Sudeste | 20 | Rio de Janeiro | | | | ... | 3 |
| 3010 | 2 | Região Sudeste | 10 | São Paulo | 101 | São Paulo | | ... | 2 |
| 3020 | 2 | Região Sudeste | 10 | São Paulo | 102 | Campinas | | ... | 2 |
| 4010 | 2 | Região Sudeste | 10 | São Paulo | 101 | São Paulo | 1001 | ... | 1 |
| 4020 | 2 | Região Sudeste | 10 | São Paulo | 102 | Campinas | 1002 | ... | 1 |
| 4030 | 2 | Região Sudeste | 10 | São Paulo | 102 | Campinas | 1003 | ... | 1 |

***Figura 8 – (a.) Tabela dimensão em detalhes com apenas uma descrição (genérica).***

***(b.) Tabela dimensão com uma descrição para cada nível da hierarquia – Star expandido.***

A Figura 8 (itens a e b) mostram os detalhes de uma tabela dimensão, denominada *Geografia,* que descreve e define uma hierarquia entre a loja, a cidade, o estado (UF) e a região onde esta se localiza. Na Figura 8a é encontrada uma das possibilidades, com apenas uma descrição genérica para toda a dimensão; já na Figura 8b colunas de descrição específicas para cada elemento da hierarquia são definidas. É importante notar que não existe uma preocupação com a normalização das tabelas no data warehouse, conforme já exposto no item 3.2.2 acima. Este é um dos aspectos que diferenciam bastante a modelagem do data warehouse da modelagem convencional. Outro aspecto a destacar é que o indicador *nível* não precisa ser numérico e pode conter o nome do nível da hierarquia, como por exemplo: “cidade”, “loja”, “mês” etc.

A dimensão *Tempo* é uma dimensão especial, sendo diferente de todas as outras dimensões. Esta dimensão requer alguns atributos comuns a todas as dimensões e três atributos especiais que serão de extrema importância no momento das consultas.

As colunas que devem compor a dimensão *Tempo* são:

- Chave única genérica, gerada pelo sistema, preferencialmente numérica e inteira, para ligação lógica com a tabela fato;
- Descrição do tempo contido naquela linha. Conforme os exemplos citados na Figura 8, esta descrição pode ser única para a tabela ou específica para cada nível da hierarquia;
- Atributos da hierarquia, como nas outras dimensões;
- Nível ou Resolução, que indica o nível da hierarquia representado naquela linha, que pode ser numérico ou um pequeno texto;
- Seqüência na resolução, que indica a seqüência de um período de tempo dentro do nível ou resolução ao qual ele pertence. Este valor é um número inteiro

consecutivo que designa a ordem cronológica para todas os dias, meses, semestres, anos etc, ou seja, para todos os períodos de tempo existentes na tabela dimensão. A seqüência é definida dentro de cada nível ou período de resolução, sem repetições e, há recomendações de que sejam utilizados intervalos de valores para estabelecer a seqüência. Por exemplo, para a seqüência de *anos* a numeração seria de 1 a 10, para *semestres*, entre 100 e 200, todos os *meses* estariam entre 500 e 999 e todos os *dias* teriam seqüências numeradas a partir de 1.000. Na Figura 9 vemos que a seqüência na resolução para Janeiro de 1999 é 500, para Fevereiro de 1999 é 501, Março de 1999 tem seqüência 502 e assim por diante. É importante ressaltar novamente que o valor para esta coluna deve ser único dentro de um período de resolução;

- Indicador Corrente, se utilizado em conjunto com a coluna de seqüência na resolução, descrita no item anterior, indica qual o período de tempo dentro da resolução especificada, que deveria ser definido como *corrente.* Com isto, é possível definir qual será o tempo corrente para as consultas: a informação carregada mais recentemente, a data de hoje ou algum período de tempo futuro. A utilização desta coluna permite que determinadas condições de negócio estabeleçam o momento em que os dados estarão disponíveis para os usuários finais. Esta coluna é, normalmente, definida como um caractere e preenchida com ‘S’ ou ‘N’, onde ‘S’ indica que o dado já foi carregado no data warehouse. O período de tempo corrente, dentro da resolução (pode ser o dia corrente, o mês corrente etc.), será aquele que contiver na coluna Indicador Corrente o valor de ‘S’ e o maior valor na coluna Seqüência na Resolução, descrita no item anterior.

| Tempo key | Descrição | Resolução ou Nível | Sequência na Resolução | Indicador Corrente | Ano | Semestre | Mês | Dia |
|---|---|---|---|---|---|---|---|---|
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 101 | Janeiro 1999 | Mês | 501 | S | 1999 | 011999 | 011999 | |
| 102 | Fevereiro 1999 | Mês | 502 | S | 1999 | 011999 | 021999 | |
| 103 | Março 1999 | Mês | 503 | S | 1999 | 011999 | 031999 | |
| 104 | Abril 1999 | Mês | 504 | S | 1999 | 011999 | 041999 | |
| 105 | Maio 1999 | Mês | 505 | S | 1999 | 011999 | 051999 | |
| 106 | Junho 1999 | Mês | 506 | S | 1999 | 011999 | 061999 | |
| 113 | Janeiro 2000 | Mês | 513 | S | 2000 | 012000 | 012000 | |
| 114 | Fevereiro 2000 | Mês | 514 | S | 2000 | 012000 | 022000 | |
| 115 | Março 2000 | Mês | 515 | S | 2000 | 012000 | 032000 | |
| 116 | Abril 2000 | Mês | 516 | S | 2000 | 012000 | 042000 | |
| 21 | 1 Sem. 1999 | Semestre | 101 | S | 1999 | 011999 | | |
| 22 | 2 Sem. 1999 | Semestre | 102 | S | 1999 | 021999 | | |
| 23 | 1 Sem. 2000 | Semestre | 103 | S | 2000 | 012000 | | |
| 24 | 2 Sem. 2000 | Semestre | 104 | S | 2000 | 022000 | | |
| 3 | 1999 | Ano | 3 | S | 1999 | | | |
| 4 | 2000 | Ano | 4 | S | 2000 | | | |
| 5 | 2001 | Ano | 5 | N | 2001 | | | |
| 6 | 2002 | Ano | 6 | N | 2002 | | | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |

***Figura 9 – Tabela representando a dimensão Tempo, com as colunas específicas para este tipo de dimensão.***

Nem sempre é possível capturar todas as métricas e dimensões importantes para um modelo de negócios, em um único esquema *Star*. Ao contrário, a existência de múltiplas tabelas fato é a norma, definindo um modelo de múltiplas estrelas. Cada estrela é composta por uma tabela fato e suas dimensões associadas.

O principal fator que pode levar à separação das métricas em diferentes tabelas fato é o modelo de negócios analisado. Um processo de negócios pode envolver diferentes conjuntos de dimensões e as métricas relacionadas a cada processo podem estar sendo coletadas em diferentes intervalos de tempo.

O esquema *Star* é desenhado para simplicidade e velocidade nas consultas. Podemos citar como vantagens, na sua utilização, os seguintes aspectos:

- Facilidade de entendimento do modelo, principalmente por parte do usuário final;
- Excelente desempenho, devido ao uso de chaves genéricas, geralmente numéricas e inteiras;
- Facilidade na definição das hierarquias;
- Número de operações de *join* reduzidas, já que o modelo implementa todos os níveis da hierarquia das dimensões em apenas uma tabela por dimensão além de estar bastante desnormalizado;
- O metadados é simples.

Na Figura 10 está representado um esquema *Star* clássico, composto por uma tabela fato e suas dimensões *Geografia*, *Produto* e *Tempo* associadas.

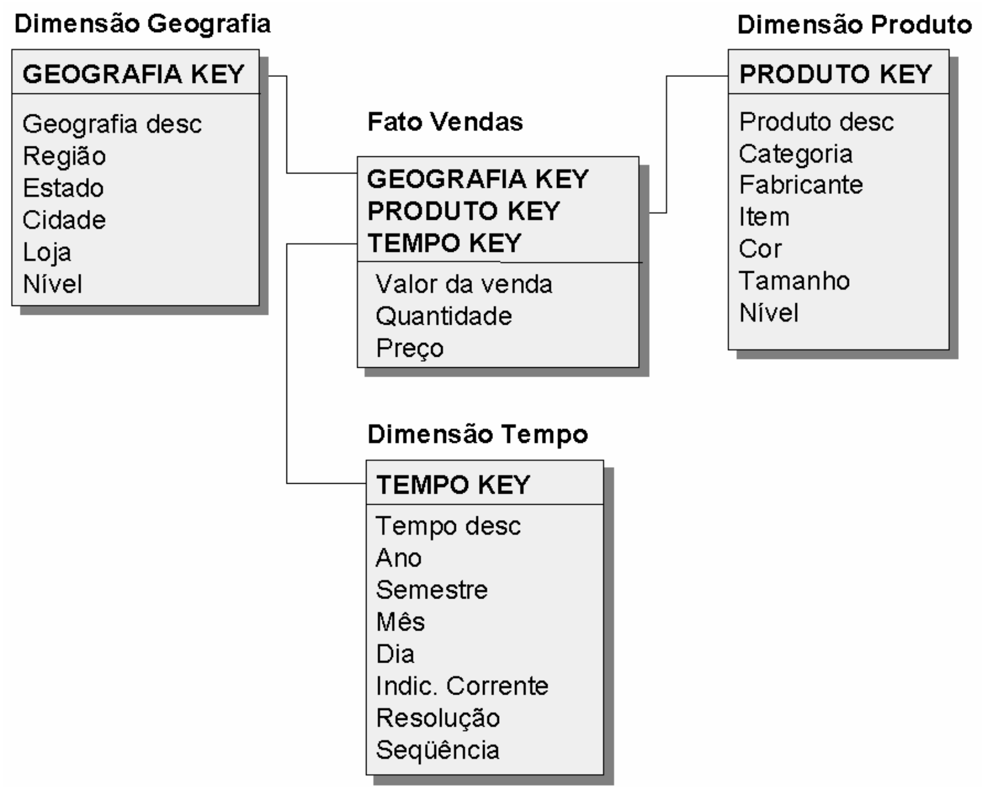



***Figura 10 – Representação de um data warehouse em esquema Star clássico***

Apesar de amplamente utilizado, o esquema *Star* apresenta alguns problemas. O primeiro deles está relacionado ao indicador *nível*, presente em todas as tabelas dimensão. Este indicador deve ser utilizado em todas as consultas, já que a mesma tabela dimensão armazena informações sobre todos os níveis da hierarquia. Durante a construção da consulta, o usuário deve conhecer a estrutura da base de dados, para poder especificar o nível da informação a ser pesquisada. Este indicador pode ser causa potencial para erros já que, se for esquecido ou mal utilizado, pode levar a resultados incorretos e baixa flexibilidade. Além disso, se a estrutura da dimensão for alterada, incluindo ou eliminando algum nível da hierarquia, o data warehouse necessitará de alterações físicas, elevando os custos de manutenção.

O segundo problema é que, por manter todos os níveis hierárquicos da dimensão fisicamente em uma única tabela, esta se torna muito grande, em vários casos. Outro fator que contribui para o aumento do tamanho da tabela é a desnormalização, que gera uma grande quantidade de valores de atributos repetidos. A desnormalização fica bastante clara, ao analisar, por exemplo, a dimensão *Geografia*, que aparece na Figura 8b. No diagrama, representado na Figura 11, podemos ver as dependências funcionais que ocorrem em uma tabela deste tipo.

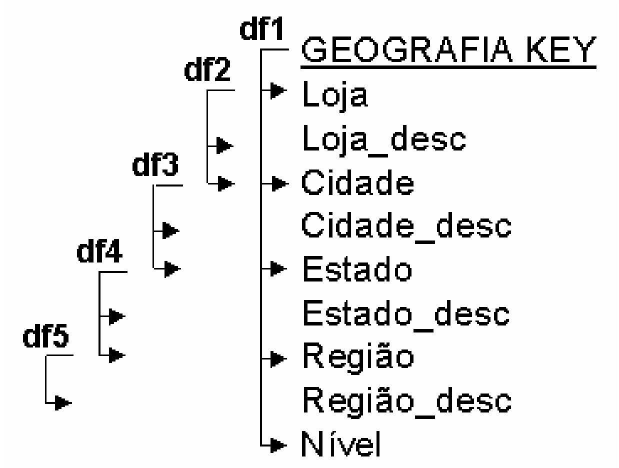



***Figura 11 – Dependências funcionais da tabela dimensão GEOGRAFIA***

O esquema *Star* clássico pode apresentar algumas variações, dependendo do problema de negócio apresentado. Nas seções seguintes seguem as variantes do *Star*.

### 10.4.1.1 As variações do esquema Star

A primeira variação a ser discutida é o esquema *Partial Star.* Nesta variação existem múltiplas tabelas para cada dimensão e para a tabela fato. Estas múltiplas tabelas estão lógica e fisicamente separadas em função dos seus níveis de agregação. Este modelo cria múltiplas estrelas, cada uma representando uma combinação de níveis de agregação em cada dimensão. Não existem ligações lógicas entre as várias tabelas fato ou dimensão, apenas entre a tabela dimensão e a tabela fato de cada grupo. Cada dimensão é representada por múltiplas tabelas que são fisicamente separadas pelo nível de agregação, sendo que as tabelas possuem como chave primária, uma chave genérica gerada pelo sistema, da mesma forma que o esquema *Star* clássico. Pode acontecer que uma métrica ocorra apenas para um determinado nível de uma dimensão. Por exemplo, a métrica *Preço* existe unicamente no nível *Item* da dimensão *Produto*. A Figura 12 retrata esta variação do *Star*, representando apenas a dimensão *Geografia*, separada nos níveis de *Região* e de *Cidade*, e as tabelas fatos associadas a estes níveis.

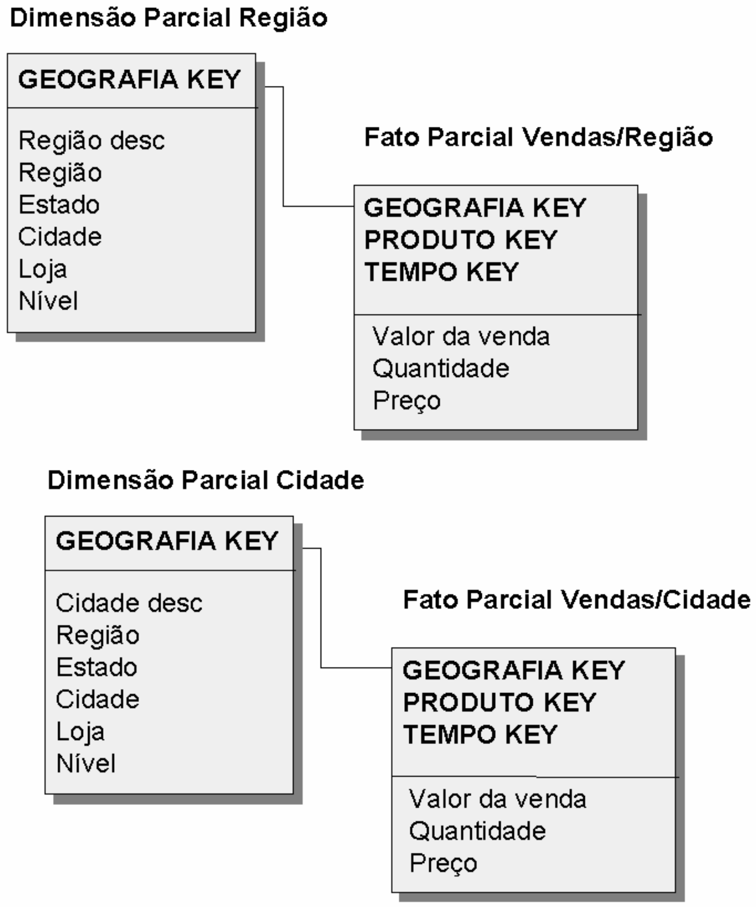



Figura 12 – Esquema *Partial Star* mostrando apenas a dimensão Geografia e as tabelas fato associadas.

Esta partição das informações, pelos níveis hierárquicos de agregação, leva à criação de duas estrelas: a primeira representando o nível de *Região* e a segunda representando o nível de *Cidade*. Se forem adicionadas a este esquema as dimensões *Tempo* e *Produto* agregadas nos níveis de *Fabricante* e *Marca*, conforme o modelo representado na Figura 10 (Star Clássico), ocorrerá uma expansão para *Região/Fabricante/Ano*, *Região/Marca/Ano*, *Cidade/Fabricante/Ano* e *Cidade/Marca/Ano*, sendo que cada combinação será representada por uma estrela parcial separada:

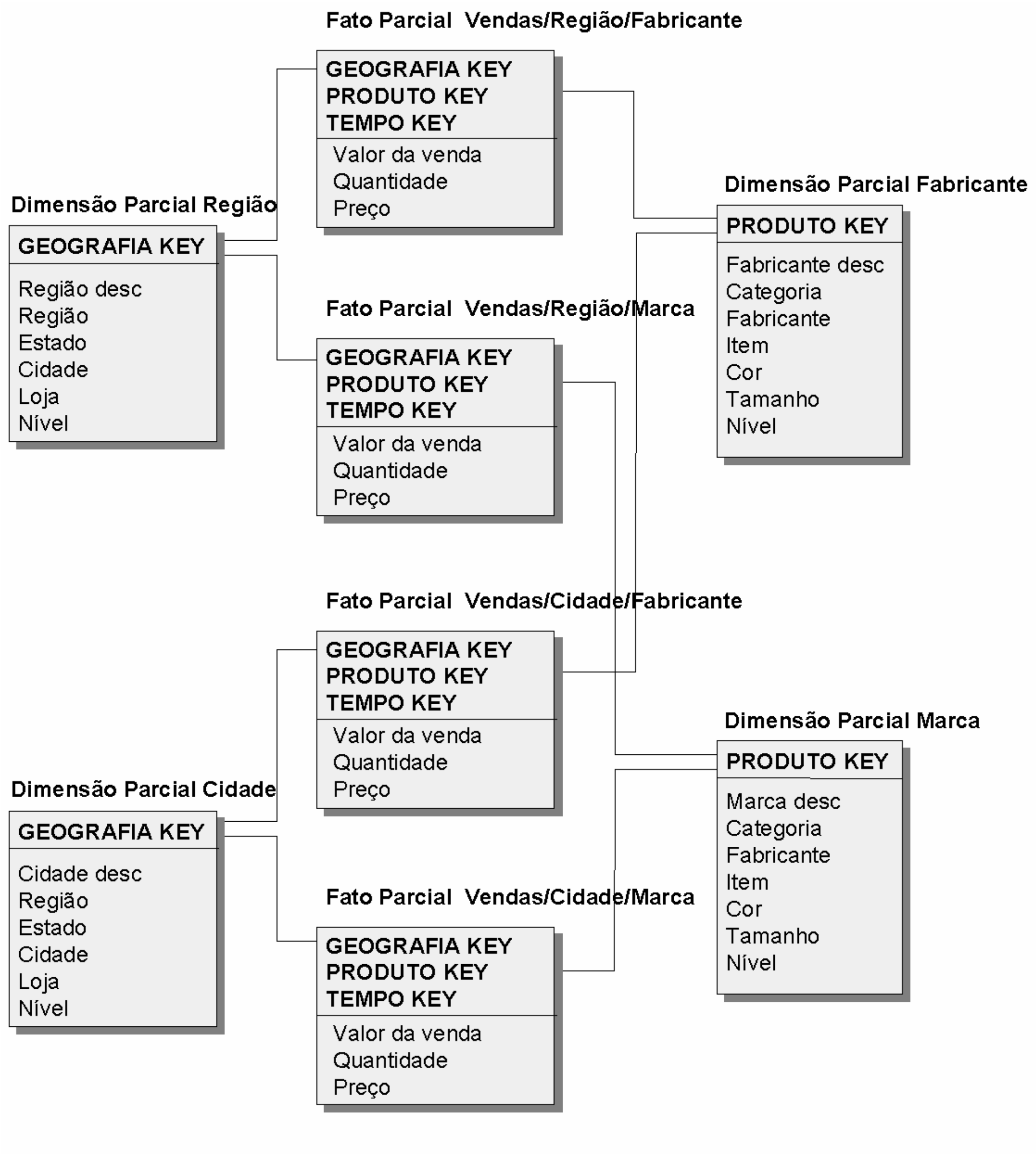


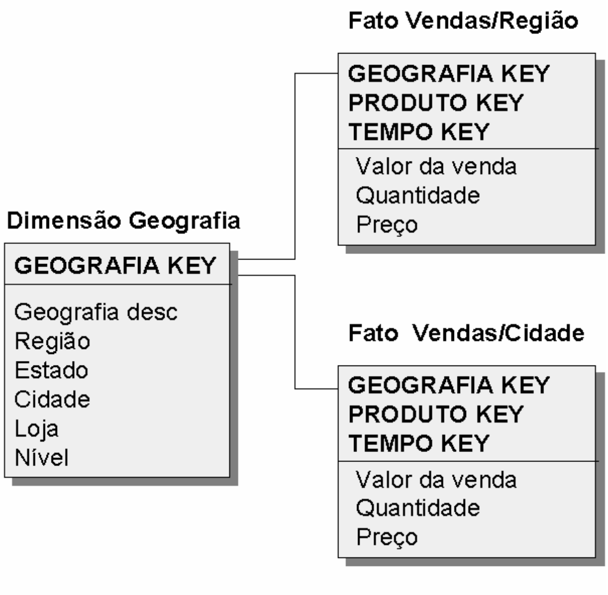

***Figura 13 – Esquema Partial Star mostrando as dimensões Geografia e Produto separadas em diferentes níveis de agregação e as tabelas fato associadas.***

São várias as vantagens encontradas no esquema *Partial Star*, apesar de sua visível complexidade. Uma delas é a possibilidade de maior controle do tempo de carga, *backup* e manutenção das tabelas, pois seu tamanho fica reduzido em função da partição. Estas partições possibilitam também a existência de fatos em certos níveis específicos e é possível eliminar as colunas que não tenham significado em determinados níveis, reduzindo assim a esparsidade. Como já dito, a complexidade do modelo aumenta em relação ao *Star* clássico e cada estrela requer uma definição específica nos metadados, dificultando o processo de manutenção.

Com relação às consultas, serão necessários múltiplos comandos SQL para analisar dados em mais de um nível de agregação em um mesmo relatório, o que pode degradar o desempenho da consulta. Além disso, a estrutura física de cada tabela ou grupo de tabelas requer alterações sempre que novos níveis de agregação ou novas combinações forem adicionados ou removidos.

Para simplificar um pouco o esquema *Partial Star*, utiliza-se uma combinação de seus princípios com o *Star* clássico. Pode-se particionar apenas a tabela fato e manter cada dimensão como uma única tabela ou o inverso, ou seja, manter a tabela fato única e particionar as dimensões. O particionamento é feito sempre baseado na agregação de determinados níveis da hierarquia. Estas variações são chamadas de *Fact Partitioning* e *Dimension* Partition, respectivamente. A variação *Fact Partitioning*, também chamada *Fact Constellation* é o esquema que particiona apenas a tabela fato. Este esquema utiliza uma única tabela para cada dimensão que fica ligada às múltiplas tabelas fato, particionadas pelos diferentes níveis de agregação. Este esquema possibilita a existência de fatos em certos níveis, já que as tabelas são particionadas, assim como o *Partial Star*. Ao particionar a informação em níveis de agregação desde o mais detalhado, para o caso de consultas executadas nos níveis de detalhe mais altos, o desempenho será melhor. Deve-se levar em conta que, na implementação deste esquema, se a hierarquia é extensa, numerosas tabelas fato serão criadas, aumentando a complexidade do desenho. Além disso, consultas que requeiram a análise das informações em mais de um nível de agregação emitirão múltiplos comandos SQL, o que pode acarretar em um desempenho inferior. A Figura 14, abaixo, representa um exemplo desta variação, onde a tabela fato está particionada em função dos níveis hierárquicos *Região* e *Cidade*, da dimensão *Geografia* (as demais dimensões não estão representadas):

***Figura 14 – Exemplo do esquema Fact Partitioning ou Fact Constellation para a dimensão Geografia e as tabelas fato correspondentes às partições por Região e Cidade.***




A variação *Dimension Partitioning* mantém uma única tabela fato, que contém as métricas dos vários níveis de agregação, incluindo o nível mais detalhado. Esta tabela fato está logicamente ligada às múltiplas tabelas de dimensão, particionadas por diferentes níveis de agregação. A Figura 15 representa um esquema deste tipo, onde a dimensão *Geografia* encontra-se particionada por *Região* e por *Cidade* (as outras dimensões não estão representadas). As vantagens da *Dimension Partitioning* incluem a possibilidade de assinalar atributos que são específicos cada nível de agregação e um bom desempenho, já que apenas um comando SQL é emitido para a tabela fato, sem a necessidade de efetuar *joins*. Ao mesmo tempo, a vantagem de manter todos os fatos, detalhados e sumariados na mesma tabela, dificulta a recuperação das informações dos níveis mais altos.

A escolha do esquema a ser implementado, deve levar em conta vários fatores do negócio, bem como os tipos de consultas e o tamanho de cada uma das tabelas definidas no data warehouse. Seja qual for a variação escolhida, o *Star* se carateriza pela simplicidade, rapidez no acesso e desnormalização. O esquema *Star* pode ser refinado e transformado em um esquema chamado *Snowflake* que, para implementar as hierarquias dos atributos, se utiliza de

tabelas de subdimensões. Estas subdimensões facilitam a normalização do modelo. Manter as tabelas desnormalizadas, implementando-se um esquema Star é um aspecto bastante discutível, pois a divisão das tabelas, como em um esquema *Snowflake*, em nome da normalização, pode levar as consultas a um desempenho mais baixo.

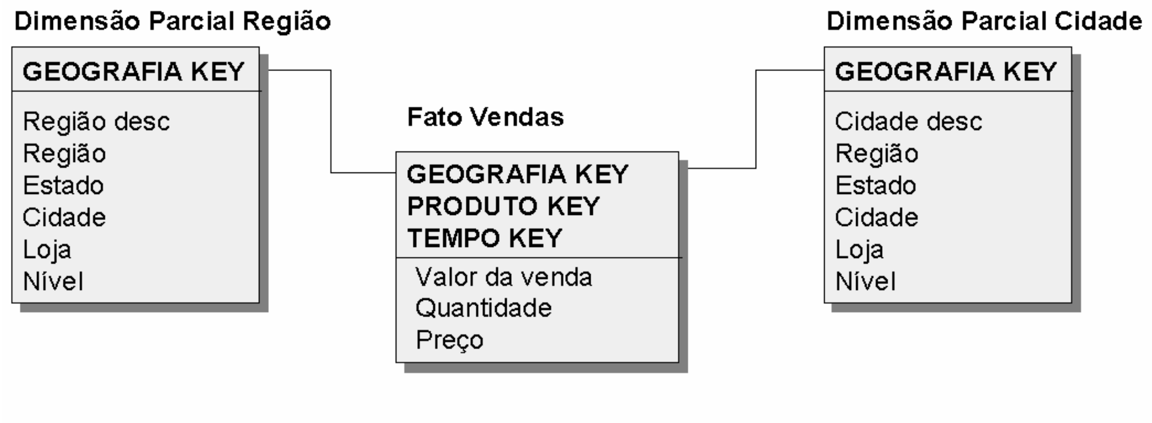



***Figura 15 – Implementação do esquema Dimension Partitioning, representando o particionamento da tabela dimensão Geografia.***

## 10.4.1.2 O esquema Snowflake

O esquema *Snowflake* pode ser considerado um *Star* normalizado, pois emprega uma combinação de normalização da base de dados, para manter a integridade e reduzir os dados armazenados de forma redundante, com uma desnormalização para obter melhor desempenho. Neste esquema as dimensões são normalizadas em subdimensões, sendo que cada nível da hierarquia fica em uma subdimensão. Por esta razão, não há necessidade de utilizar o indicador de nível que existe nos esquemas do tipo *Star*. A tabela principal da dimensão tem uma chave para cada nível hierárquico representado na subdimensão e não mais uma única chave, como no *Star.*

O *Snowflake* apresenta duas variações básicas que diferem na disposição das tabelas que representam as subdimensões, os *Snowflake Lookup* e o *Snowflake Chain*, que serão descritos na próxima seção. Sua representação gráfica fica similar a um floco de neve, devido ao particionamento das tabelas dimensão.

## 10.4.1.3 As variações do esquema Snowflake

O esquema *Snowflake Lookup* emprega tabelas adicionais para nomes e descrições dos atributos, todas ligadas a uma tabela principal da dimensão. Desta forma é possível reduzir o tamanho da tabela dimensão, eliminando a redundância do armazenamento das mesmas descrições em várias linhas diferentes, sendo que as tabelas adicionais atuam como tabelas de *lookup* para a chave ou valores codificados da tabela principal da dimensão que, por sua vez, está logicamente ligada a uma única tabela de fatos. A Figura 16 mostra o mesmo exemplo citado na Figura 10, porém, modelado em *Snowflake Lookup*, sendo representada apenas a dimensão *Geografia* e suas subdimensões *Região*, *Estado*, *Cidade* e *Loja*.

Na Figura 16, pode-se notar que a ligação entre a tabela fato e a tabela da dimensão principal é feita da mesma forma que no esquema *Star*, por meio de uma chave genérica gerada. A tabela principal da dimensão se conecta logicamente às subdimensões, que são as tabelas de *lookup*, através de suas chaves primárias.

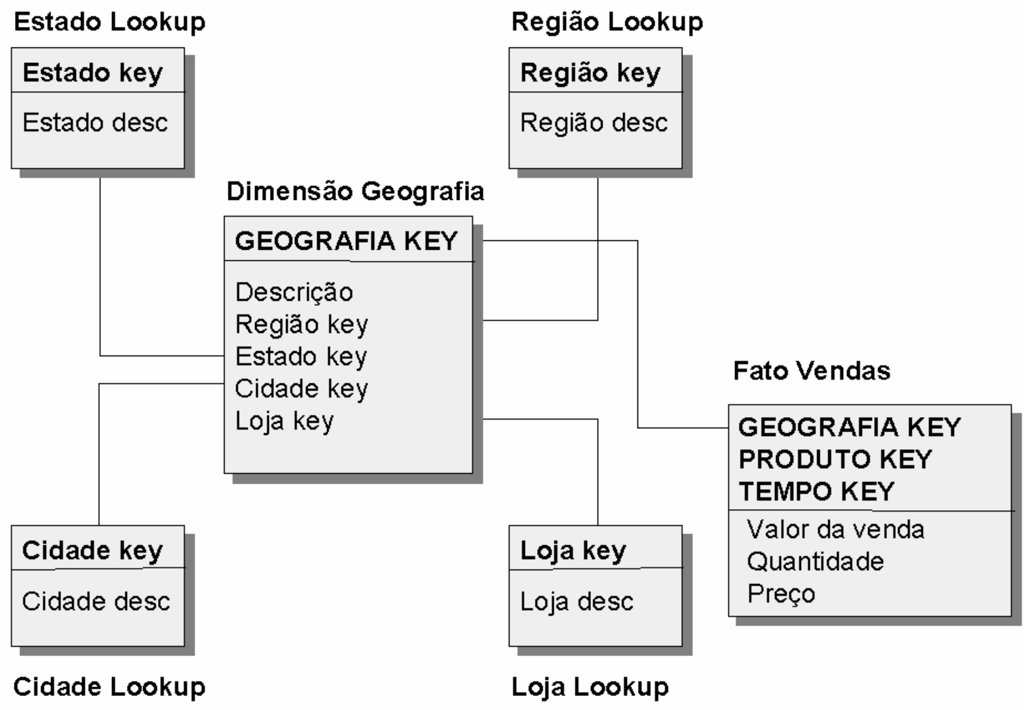



***Figura 16 – Desenho lógico da dimensão Geografia em um data warehouse implementando o esquema Snowflake Lookup***

As descrições não precisam ser repetidas como no esquema *Star*, simplificando o armazenamento, reduzindo o tamanho relativo das tabelas dimensão e melhorando o controle de integridade dos dados. O uso das chaves geradas genéricas, geralmente números inteiros, levam a um melhor desempenho, menor manutenção do metadados e mais flexibilidade ao data warehouse. Vale ressaltar que o desempenho pode ficar afetado se múltiplas consultas ou múltiplos *joins* têm que ser emitidos para decodificar as chaves, ao buscar as descrições nas tabelas adicionais. Além disso, a manutenção da base de dados requer manutenção adicional dado o maior número de tabelas físicas distintas.

O esquema *Snowflake* apresenta um nível muito maior de normalização, se comparado aos esquemas do tipo *Star*. A Figura 17 mostra os diagramas que representam as dependências funcionais do esquema que aparece na figura 16 em *Snowflake Lookup.*

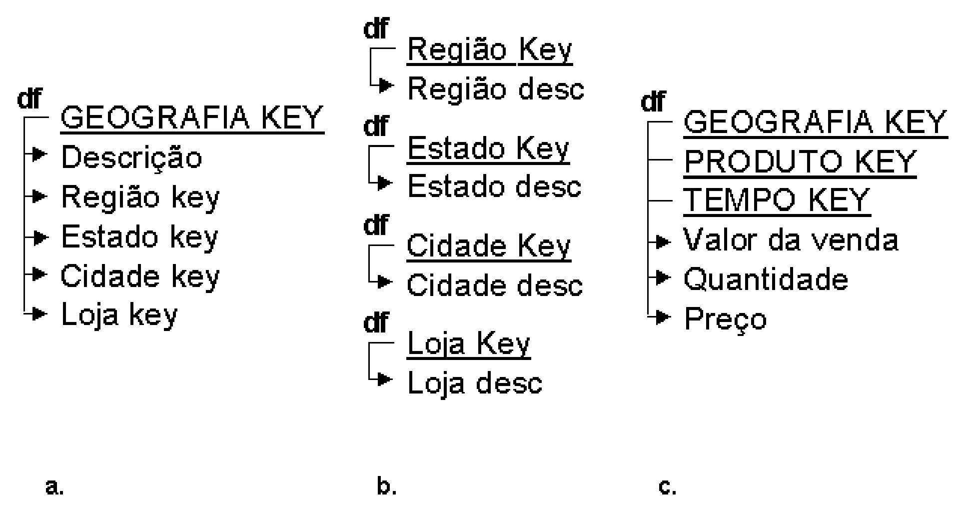



***Figura 17 – Diagramas representando as dependências funcionais em:***
***a. Tabela principal da dimensão Geografia;***
***b. Tabelas de Lookup (subdimensões) e***
***c. Tabela Fato Vendas***

A segunda variação do esquema *Snowflake*, conhecida por *Snowflake Chain*, utiliza também subdimensões, particionadas pelos níveis hierárquicos da dimensão, sendo que a tabela principal da dimensão representa o nível mais baixo (mais detalhado) da hierarquia. As subdimensões estão encadeadas, partindo-se da tabela fato que está logicamente conectada à tabela da subdimensão principal ou raiz. Na Figura 18 encontra-se um exemplo desta implementação, representando a dimensão *Geografia* com suas subdimensões e a tabela fato de vendas.




***Figura 18 – Exemplo de implementação do esquema Snowflake Chain, representando a ligação da tabela Fato com a dimensão Geografia (dividida em subdimensões encadeadas).***

Cada tabela da subdimensão contém sua chave primária e suas descrições associadas. Além disso, contém também a chave para o próximo nível da hierarquia da dimensão e assim por diante, até chegar à última subdimensão, que contém o nível mais alto (menos detalhado) da hierarquia. É tipicamente utilizada quando os fatos contêm informações apenas sobre o nível de detalhe mais baixo da hierarquia. Como verificado pela Figura 18, a tabela fato já não utiliza uma chave para a dimensão *Geografia* como um todo, a chave utilizada é diretamente para a subdimensão principal. Fica claro que esta implementação não é recomendada quando os relatórios necessitam freqüentemente vários níveis de agregação da informação, já que são necessários vários passos na cadeia para se chegar ao resultado. Este esquema oferece um alto grau de integridade de dados, pois os nomes e as descrições são mantidos em um único local, reduzindo o tamanho das tabelas de dimensão.

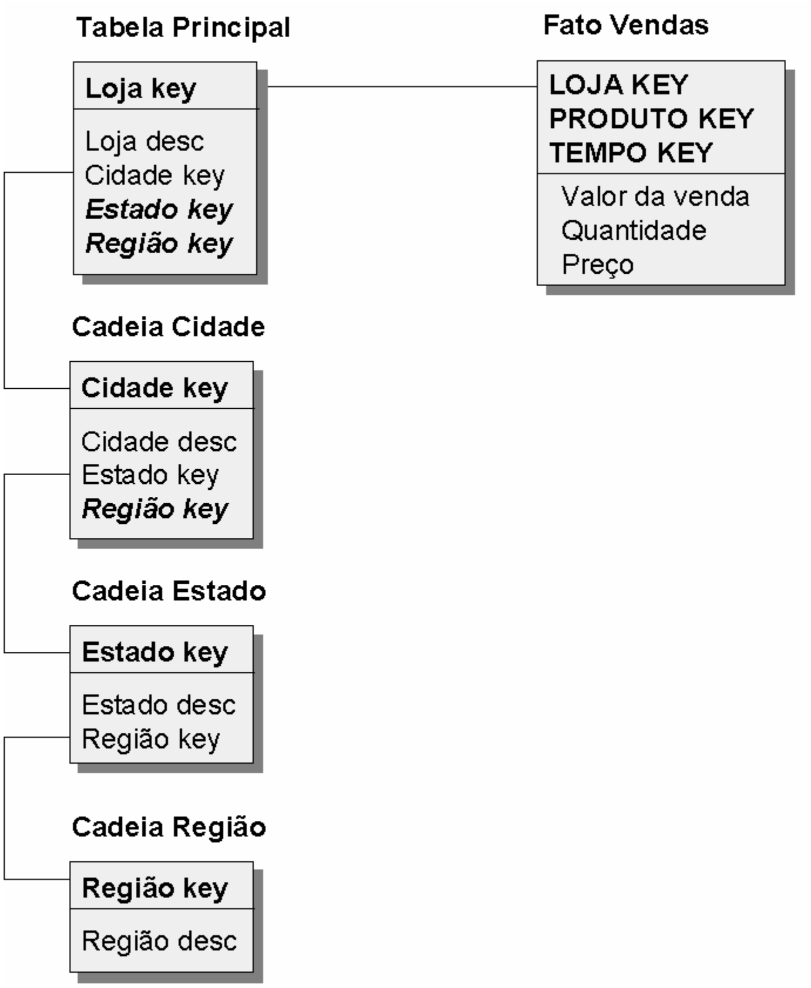



***Figura 19 – Exemplo de Snowflake Chain com chaves adicionais para melhorar o desempenho das consultas.***

A maior desvantagem do *Snowflake Chain* está no baixo desempenho, uma vez que todos os níveis da cadeia devem ser acessados, mesmo quando se requer apenas os níveis mais altos de agregação. Em situações práticas, existe uma alternativa para minimizar este efeito, adicionando-se, a cada subdimensão, chaves para todos os níveis superiores da hierarquia. Com isso, pode-se “saltar” determinados níveis, utilizando-se a chave que leva diretamente para o nível requerido. A Figura 19 apresenta esta alternativa, que visa basicamente um melhor desempenho do *Snowflake Chain*.

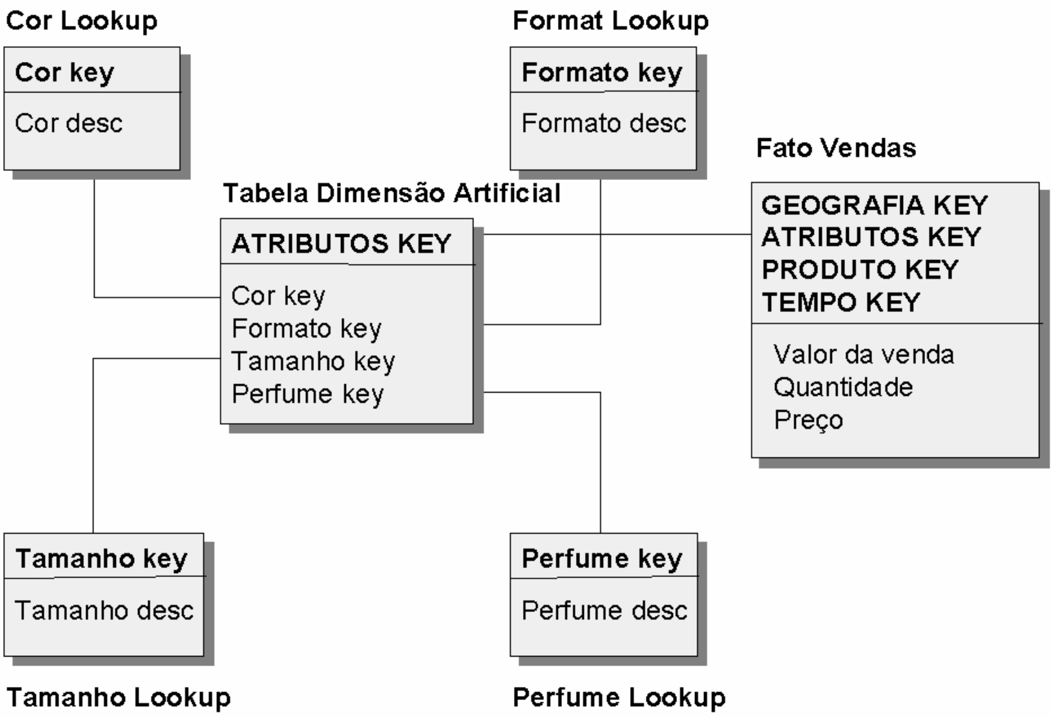



***Figura 20 – Exemplo do esquema Snowflake Attribute agrupando os atributos Cor, Formato, Tamanho e Perfume, características dos produtos vendidos.***

Podem ocorrer, em determinados modelos, vários atributos diferentes que não estão associados a nenhuma dimensão específica. Em vez de criar várias dimensões com apenas um atributo, é possível agrupá-los em apenas uma dimensão. Este esquema é chamado de *Snowflake Attribute*, sendo que a dimensão artificial pode também implementar a combinação de dimensões pouco utilizadas com o objetivo de reduzir o número de dimensões do data warehouse, simplificar o modelo e melhorar o desempenho. A tabela principal desta dimensão artificial contém as chaves estrangeiras dos diferentes atributos ou das tabelas dimensão a serem agrupadas. A chave primária da dimensão artificial é uma chave genérica gerada pelo sistema e uma única linha é adicionada para cada combinação válida de todos os atributos ou dimensões envolvidas. Cada chave estrangeira da tabela principal leva a uma chave primária na tabela de descrição daquele atributo ou dimensão. A Figura 20 traz um exemplo, onde *Tamanho*, *Cor*, *Formato* e *Perfume* são atributos não dimensionais dos produtos vendidos, que não podem ser associados a nenhuma das dimensões existentes no modelo. Utilizando-se este tipo de esquema, os atributos podem ser agrupados em uma única dimensão artificial que contém todas as combinações válidas destes atributos, sendo que a tabela fato contém uma chave para conectar-se logicamente a esta dimensão, da mesma forma que se conecta às outras dimensões do modelo.

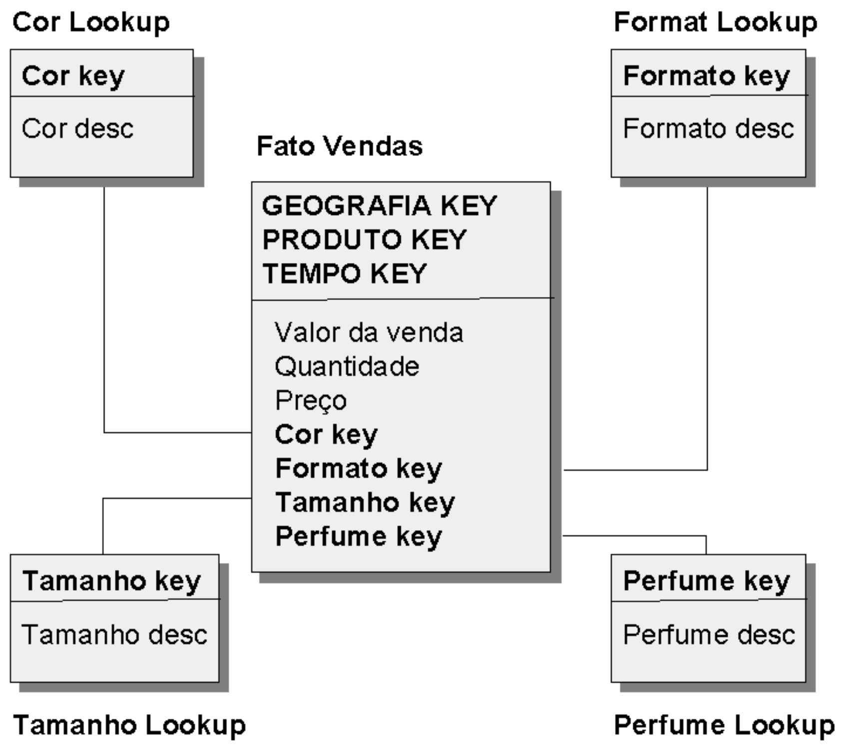



***Figura 21 – Implementação alternativa sem a utilização do esquema Snowflake Attribute.***

As vantagens encontradas neste tipo de implementação são várias: redução do número de dimensões, que leva à redução do tamanho global dos índices e da complexidade do SQL gerado, maior integridade dos dados e os vários atributos juntos ou dimensões pouco usadas compartilham de um único ponto de entrada na tabela de fatos.

Sem este tipo de esquema, cada um destes atributos ou dimensões seria mantido como uma chave estrangeira na tabela fato, resultando em um ponto de entrada que não faria parte da chave primária e, provavelmente, sem índice físico para facilitar o acesso. Esta situação, retratada na Figura 21, deve ser evitada.

## 10.5 Agregações das informações

Apesar dos dados no data warehouse serem armazenados segundo a granularidade definida, muitas das consultas realizadas necessitam, além das informações detalhadas, de informações sumariadas ao longo das dimensões. A informação armazenada no nível de detalhe é importante, porém o acesso à informação em níveis sumariados permite aos analistas de negócio terem uma visão global do modelo de negócios analisado. Estas consultas, partindo de uma base onde existem apenas os dados de nível básico, ou seja, do nível mais detalhado, se for necessário sumariar os dados no momento da execução, todo o processo de análise será sobrecarregado. Um determinado conjunto de vários agregados pré-computados faz-se necessário para acelerar cada uma das consultas, sendo que o efeito sobre o desempenho é considerável, obtendo reduções drásticas no tempo de processamento. Por esta razão, a utilização de dados previamente sumariados (agregados) é um recurso bastante eficiente para controlar o desempenho do data warehouse.

### 10.5.1 Definindo os agregados

A escolha dos agregados a serem previamente armazenados no data warehouse, já que armazenar todos os agregados é impraticável, não é uma tarefa trivial. É necessária uma

análise das consultas que serão executadas pelos usuários, identificando os agrupamentos mais freqüentes, além de constante acompanhamento após a implantação.
Uma forma de descrever as agregações a serem aplicadas na base é como um "agregado *n*-direcional", onde *n* indica o número de dimensões sumariadas como resultado da agregação. Um exemplo de um agregado *uni*-direcional seria sumariar uma dimensão (em algum nível da hierarquia) enquanto que as outras dimensões ficam em seu nível atômico, mais detalhado. Um agregado *bi*-direcional sumaria um nível da hierarquia para duas dimensões deixando as demais em seu nível atômico. Utilizando o exemplo expresso na Figura 10, que representa um esquema *Star* clássico com três dimensões – *Geografia*, *Produto* e *Tempo* e, quiser implementar no data warehouse exemplo, informações sumariadas para totais de *Categoria* na dimensão *Produto*, totais de *Estado* na dimensão *Geografia* e totais mensais na dimensão *Tempo*, ter-se-á sete tipos de agregados diferentes:

1. Agregado unidirecional: totais de *categoria* por *loja* por *dia*;
2. Agregado unidirecional: totais de *cidade* por *item de produto* por *dia*;
3. Agregado unidirecional: totais mensais por *item de produto* por *loja*;
4. Agregado bidirecional: totais de *categoria* por totais de *cidade* por *dia*;
5. Agregado bidirecional: totais de *categoria* por totais mensais por *loja*;
6. Agregado bidirecional: totais de *cidades* por totais mensais por *item de produto*;
7. Agregado tridirecional: totais de *categoria* por totais de *cidade* por totais mensais.

O exemplo evidencia a não representação de todos os possíveis agregados para as dimensões do modelo, o que seria impraticável, dada a grande quantidade de dados a ser administrada. Ainda assim, para os agregados escolhidos é necessária uma avaliação do tamanho do resultado a ser gerado, levando-se em conta a explosão do número de linhas e a dispersão das informações armazenadas no data warehouse. Uma tabela de fatos de nível básico, para o exemplo em tela, é normalmente bastante dispersa no preenchimento das chaves. Para entender este aspecto, lembre-se que apenas uma parte dos produtos é vendida por dia em cada uma das lojas. Porém, ao calcular o tamanho do conjunto de agregados, deve-se considerar que sua criação aumenta drasticamente a taxa de ocupação, aumentando substancialmente o tamanho do data warehouse. Para comprovar este fato, observe um exemplo prático em um data warehouse fictício. Suponha que no modelo exemplo em análise existem 10.000 itens produtos diferentes, 1.000 lojas e o período de tempo armazenado é de 2 anos, equivalente a 730 dias. Além disso, neste data warehouse exemplo, apenas aproximadamente 10 porcento dos produtos são vendidos em uma determinada loja em um determinado dia. Isto significa que a tabela de fatos ocupa apenas 10 porcento do que ocuparia caso todos os produtos fossem vendidos em todas as lojas, todos os dias. Entretanto, com a criação dos agregados esta taxa aumenta consideravelmente. Suponha ainda que, no caso de produtos, 10.000 itens levam a 2.000 categorias de produto e que as 1.000 lojas estão agrupadas em 100 cidades e o período de tempo total será sumariado em 24 períodos mensais. Além disso, vamos considerar a dispersão e o número de linhas a tabela de fatos, de acordo com a tabela apresentada a seguir:

| **Tabela** | **Produto** | **Geografia** | **Tempo** | **Dispersão** | **# Linhas** |
|---|---|---|---|---|---|
| Base: item de produto por loja por dia | 10.000 | 1.000 | 730 | 10% | 730 milhões |
| Unidir.: categoria por loja por dia | 2.000 | 1.000 | 730 | 50% | 730 milhões |
| Unidir.: item por cidade por dia | 10.000 | 100 | 730 | 50% | 365 milhões |
| Unidir.: item por loja por mês | 10.000 | 1.000 | 24 | 50% | 120 milhões |
| Bidir.: categoria por cidade por dia | 2.000 | 100 | 730 | 80% | 116 milhões |
| Bidir.: categoria por loja por mês | 2.000 | 1.000 | 24 | 80% | 38 milhões |
| Bidir.: item por cidade por mês | 10.000 | 100 | 24 | 80% | 19 milhões |
| Tridir.: categoria por cidade por mês | 2.000 | 100 | 24 | 100% | 4,8 milhões |
| | | | | **Total aproximado:** | 2.122 milhões |

Se considerarmos o tamanho inicial da tabela de fatos base, que é de aproximadamente 730 milhões de linhas, e o número de linhas após a criação destes sete níveis de agregados,

verificamos que o aumento foi de 200%. O fator de dispersão pode ser difícil de prever e a escolha dos agregados deve ser feita com bastante cuidado. Ainda assim, a forma mais eficiente para controlar a explosão de agregados, mas ainda assim beneficiar-se de seu valor, é garantir que, em média, cada agregado resuma no mínimo 20 e, de preferência, 20 ou mais itens de nível básico.

É importante lembrar que um determinado agregado pode servir para acelerar uma consulta que requeira um agregado de nível superior, ou seja, se existe um agregado de nível intermediário na hierarquia, este pode ser utilizado, não sendo necessário executar a consulta agregando o nível mais detalhado.

### 10.5.2 Implementando os agregados

A implementação dos valores agregados no data warehouse conta com várias opções de desenho. É possível manter os fatos sumariados na mesma tabela fato original ou separados por tipo de agregado, o mesmo ocorre com as dimensões. A forma de se implementar os agregados vai depender do esquema implementado para o data warehouse e do número de linhas geradas pelos agregados.

Para um esquema do tipo *Star*, onde para cada categoria de informações é gerada uma estrela composta por uma tabela fato e uma tabela para cada dimensão, os agregados podem residir na tabela de fatos original e seus identificadores já estão presentes nas tabelas de dimensão. Cada linha da tabela fato é identificada por uma chave composta, como já visto anteriormente, que identifica, nas dimensões, os elementos associados aos valores armazenados na fato. Antes da criação dos agregados, a tabela fato contém apenas valores associados ao nível mais detalhado de cada dimensão. Assim, com a inclusão dos valores agregados na tabela fato, estes deverão relacionar-se com os outros níveis da hierarquia das dimensões. Isto é possível, já que cada tabela dimensão, no esquema *Star*, possui uma linha descrevendo cada elemento da hierarquia, sendo que o campo *Nível* identifica esta estrutura hierárquica. É claro que, se os agregados são representados nas tabelas de dimensão e fatos originais, por meio desta estrutura de campos *Nível*, todas as consultas feitas a esse esquema devem restringir o campo *Nível* a um único valor, caso contrário, ocorrerá contagem dupla, envolvendo valores relacionados a níveis de agregação diferentes. A Figura 21 mostra este relacionamento, representando uma dimensão *Geografia* e uma tabela fato que possui valores para os vários níveis da dimensão.

**Tabela Fato:**

| | Tempo key | Produto key | Geografia key | Valor | Quantidade |
|---|---|---|---|---|---|
| (1) | 980715 | 101 | 4030 | 348,00 | 140 |
| (2) | 980716 | 101 | 4030 | 287,00 | 114 |
| (3) | 980716 | 101 | 4010 | 443,00 | 170 |
| (4) | 980716 | 102 | 4010 | 630,00 | 158 |
| (5) | 980717 | 101 | 2010 | 5.800,00 | 2320 |
| (6) | 980718 | 102 | 2010 | 7.540,00 | 1720 |
| (7) | 980718 | 101 | 2010 | 6.860,00 | 2740 |
| (8) | 980716 | 101 | 3010 | 2.840,00 | 1184 |
| (9) | 980716 | 103 | 3010 | 5.834,00 | 4059 |
| | ... | ... | ... | ... | ... |

**Tabela Dimensão Geografia:**

| Geografia key | Geografia Descrição | Região | Estado | Cidade | Loja | Nível |
|---|---|---|---|---|---|---|
| 1010 | Região Sul | 1 | | | | Região |
| 1020 | Região Sudeste | 2 | | | | Região |
| 2010 | SP | 2 | 10 | | | Estado |
| 2020 | RJ | 2 | 20 | | | Estado |
| 3010 | São Paulo | 2 | 10 | 101 | | Cidade |
| 3020 | Campinas | 2 | 10 | 102 | | Cidade |
| 4010 | Loja ABC | 2 | 10 | 101 | 1001 | Loja |
| 4020 | Loja DEF | 2 | 10 | 102 | 1002 | Loja |
| 4030 | Loja GHI | 2 | 10 | 102 | 1003 | Loja |
| ... | ... | ... | ... | ... | ... | ... |

Figura 22 – Representação de uma única tabela fato contendo informações de nível detalhado e de agregados associados à dimensão Geografia.

Como podemos verificar, na Figura 22, a tabela fato continua sendo única, contendo as linhas de fatos para os níveis detalhados e agregados da dimensão. No exemplo, foram representados parcialmente dois agregados unidirecionais, sendo:

- Agregado: *Item de produto* por *dia* por *Estado* e
- Agregado: *Item de produto* por *dia* por *Cidade*.

Portanto, na tabela fato acima, as linhas (1), (2), (3) e (4) representam valores associados aos níveis detalhados *Item de produto*, *Dia* e *Loja*, enquanto que as linhas (5), (6), (7), (8) e (9) representam os agregados.

A grande desvantagem desta implementação é a utilização do campo *Nível* que, como citado anteriormente, pode acarretar contagem dupla, se não for utilizado corretamente.

Uma alternativa seria manter os fatos agregados em tabelas distintas, ou seja, uma tabela de fatos para cada agregado. Como vantagens desta alternativa de implementação pode-se citar:

- Não ocorrerá contagem dupla em qualquer aplicação que utilize as tabelas;
- Os vários tipos de agregados podem ser criados, eliminados, carregados e indexados separadamente quando estão em tabelas diferentes. Como eles provavelmente serão introduzidos no banco de dados em momentos diferentes, o uso de tabelas separadas permite um gerenciamento incremental.

Em um esquema do tipo *Snowflake*, a implementação mais adequada é, obviamente, a utilização de tabelas fato separadas por agregado, já que as dimensões são também separadas. Não há motivos para implementar o atributo *Nível* em um esquema do tipo *Snowflake* unicamente para manter os agregados em apenas uma tabela fato.

## 10.6 Utilizando os agregados com um novo componente: o navegador de agregados

A melhor alternativa para a utilização dos agregados é, sem dúvida, o pré-armazenamento dos agregados mais utilizados, sendo que algumas vezes a agregação, por não existir um agregado compatível, é efetuada pela consulta durante sua execução.
O controle deste procedimento não deve ficar a cargo do usuário final, pois a criação ou eliminação de algum agregado levaria a uma série de procedimentos administrativos impraticáveis. Por esta razão, em um data warehouse projetado corretamente, os usuários finais e desenvolvedores de aplicações jamais verão quaisquer dessas tabelas de agregados. Torna-se necessária, portanto, a definição de um componente de software que escolha a tabela de agregados mais apropriada durante uma consulta, baseado, normalmente, em informações existentes na camada de metadados do data warehouse. Assim, este componente chamado por R. Kimball de navegador e em em outras fontes bibliográficas de *Aggregate Aware*, para citar apenas dois exemplos, é capaz de criar as combinações de agregação, que não estejam presentes no data warehouse, de forma dinâmica durante a execução da consulta.
É importante que este componente seja capaz de criar estas agregações dinâmicas partindo do agregado predefinido mais adequado para a consulta e otimizando ao máximo o desempenho destas agregações.
A utilização deste navegador não elimina as preocupações relacionadas aos agregados. Como estas agregações dinâmicas são mais lentas que o acesso a um agregado armazenado, recomenda-se uma análise profunda para a definição destes agregados, seguida de acompanhamento ininterrupto de sua utilização.

### 10.6.1 O processo de carga

Uma das leis imutáveis do data warehousing é sobre a complexidade do processo de carga. Vários aspectos contribuem para aumentar a complexidade desta tarefa: baixa qualidade e ausência de dados adequados, péssima documentação dos sistemas, múltiplas, redundantes e fontes de dados conflitantes. Muitas características complexas dos dados ficam "ocultas", vindo à tona somente no momento do processo de carga do data warehouse. A utilização de ferramentas que automatizam este processo pode facilitar o trabalho, porém a complexidade semântica envolvida no processo de transformação dos dados não pode ser minimizada. O processo de carga é, na grande maioria das vezes, redundante e desnecessariamente complicado, tornando-se um ponto crítico para a introdução de erros.
A complexidade desta etapa surge devido à diversidade e abrangência das tarefas envolvidas, recuperando, convertendo e migrando os dados a partir das diversas fontes, executando sua transformação de modo a garantir que a base de dados resultante compreenda sempre informações precisas, integradas, válidas e disponíveis em tempo hábil.
O processo de extração conta com várias dificuldades operacionais como, por exemplo, encontrar uma janela de tempo adequada para a execução do conjunto de programas necessários para o processo, normalmente em *batch*, principalmente se a atualização do data warehouse for diária. Outro problema a ser enfrentado refere-se ao ponto de sincronismo entre os vários sistemas transacionais, que são atualizados em diferentes intervalos de tempo e que servirão de entrada para este processo de extração.
Uma vez que a periodicidade do processo de carga esteja definida, é necessário escolher um método para **identificar e capturar as informações que foram atualizadas** nas bases de origem desde a execução da última carga. Existem algumas opções que podem ser empregadas para esta tarefa, podendo citar como as mais importantes:

- Utilização de *Logs*: a maioria dos gerenciadores de bancos de dados mantém *logs* ou *journals* que podem ser utilizados para identificar as alterações ocorridas nas bases de dados. Estas alterações, uma vez identificadas, devem ser escritas em um arquivo diferente que será utilizado para migrar as alterações para o ambiente de data warehouse. Esta alternativa produz bons resultados se as atualizações a serem capturadas são obtidas de forma simples e direta. Porém, se os dados necessários para a atualização do data warehouse estiverem em um número considerável de arquivos ou bases de dados separados, dificultando o sincronismo das *logs*, este processo começa a aumentar em complexidade. Adicionalmente,

pode-se citar os problemas relativos às alterações necessárias no programas de carga devido a novas versões dos gerenciadores de bancos de dados que podem levar a modificações na estrutura dos *logs*.

- Comparação das informações: apesar de conceitualmente simples, é uma opção bastante trabalhosa para executar. Consiste em desenvolver um programa que leia os dados nas bases fonte e no data warehouse, identificando as alterações ocorridas desde a última execução da carga. Este método é caro, no que se refere aos recursos de máquina necessários para comparação dos dados e pode ser muito demorado, dependendo do tamanho das bases pesquisadas e dos recursos disponíveis para o processamento.
- Reengenharia dos sistemas aplicativos transacionais: esta opção não se mostra muito atrativa, já que requer que as aplicações em produção sejam alteradas com o objetivo de armazenar, em arquivos destinados para este fim, as modificações ocorridas nos dados fonte. Ocorre, porém, que a maioria das aplicações não pode sofrer facilmente um processo de reengenharia, sendo que esta opção, em muitas situações, só poderia ser empregada nas novas aplicações a serem desenvolvidas. Neste caso, as necessidades do data warehouse passariam a fazer parte da especificação da aplicação.

Após sua captura, os dados devem passar por uma etapa de **transformação**, já que, simplesmente obtê-los do ambiente fonte e passá-los para o data warehouse não é suficiente. É necessário, com o objetivo de otimizar o potencial do data warehouse, transformar as entidades obtidas em novas composições de entidades requeridas para processo de geração de informação relevante para o usuário.

Além da transformação, os dados passam por um processo de **enriquecimento**, que é, normalmente, um produto da integração de dados e ocorre quando um atributo adicional é acrescentado a uma entidade. Se um dado externo está sendo acrescentado ao data warehouse, uma entidade *Cliente*, por exemplo, pode ser "enriquecida" com este novo atributo que foi selecionado de fontes econômicas, demográficas, financeiras, etc. Estas fontes podem ser valiosas para o "enriquecimento" das informações, porém mais valiosas ainda, podem ser as próprias aplicações do data warehouse cujos atributos podem ser derivados dos padrões de comportamento analisados.

Uma outra etapa necessária à migração dos dados envolve um **mecanismo de transporte** partindo do ambiente operacional para o data warehouse. Estes dois ambientes podem estar localizados em distintas plataformas, além da possibilidade de estarem fisicamente remotas levando a uma maior dificuldade na transferência dos dados. Por ser inconcebível um meio de transporte de dados que não seja eletrônico, a implementação do mecanismo de transporte pode envolver uma avaliação de todas as opções disponíveis, já que diferentes fabricantes podem oferecer diferentes opções proprietárias com custos, capacidades e limitações bastante distintas. Ao considerar a velocidade da transferência, deve-se levar em conta cada aspecto envolvido no processo, seja na plataforma de origem ou na plataforma destino. Esta análise é particularmente importante para determinar, no contexto considerado, o tempo disponível para a execução desta etapa nas plataformas envolvidas.

Na seqüência do processo de carga um fator muito importante está relacionado a garantir a **integridade dos dados** que estão sendo adicionados ao data warehouse. Este controle de integridade deve ser implementado de forma a garantir dois aspectos. O primeiro deles é a garantia de que os dados extraídos dos sistemas de origem são exatamente os mesmos que estão sendo carregados no data warehouse. O segundo aspecto deve garantir que os dados que estão sendo carregados estão consistentes com os dados que foram pedidos para as bases de origem em um determinado instante no tempo.

A **reformatação dos dados** pode também ser uma etapa necessária neste processo, uma vez que os ambientes de origem e o data warehouse podem estar em ambientes computacionais heterogêneos, que muitas vezes causa problemas relacionados ao formato dos dados. O problema mais comumente encontrado refere-se às diferenças entre os formatos de arquivos em EBCDIC, encontrados nos ambientes IBM e os formatos ASCII utilizados pela maioria das outras plataformas.

Como último passo do processo, encontramos a etapa de **carga dos dados** que depende, obviamente, do volume de dados envolvido no processo. Pode ser que o tempo necessário para a carga dos dados adicionado ao tempo requerido para transferi-los, provoque um impacto negativo na disponibilidade do data warehouse para os usuários. Portanto, devem ser

considerados todos os aspectos que possam dificultar este processo. Um destes aspectos é a quantidade de índices criados nas tabelas do data warehouse. Uma grande quantidade de índices adequados agiliza a execução das consultas, objetivo primário de um data warehouse, no entanto, este alto nível de indexação pode diminuir sensivelmente a velocidade do processo de carga.

É fácil perceber, após uma análise das etapas necessárias ao processo de carga, a alta complexidade desta tarefa que, se não for definida com bastante cuidado, pode levar à introdução de erros, comprometendo todo o processo de tomada de decisão executado pelos usuários. Além disso, qualquer alteração na arquitetura dos sistemas transacionais e nas bases de origem deve ser detalhadamente verificada, já que pode causar alterações em várias etapas do processo de carga.

# APÊNDICE A

Em construção.

Para informações provisórias acesse:

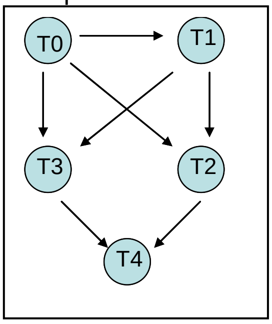
http://www.ime.usp.br/~jef/RDBLang/

# APÊNDICE B

Exercícios Processamento de transações e controle de concorrência

1) Defina o conceito de atomicidade de transações e suas propriedades.

2) Considere o grafo de precedência da figura abaixo. É possível encontrar um escalonamento seriável e seus escalonamentos seriais equivalentes? Justifique sua resposta.




3) Quais dos seguintes escalonamentos são seriáveis? Para cada escalonamento seriável determine os escalonamentos seriais equivalentes.
*a. r1(X); r3(X); w1(X); r2(X); w3(X);*
*b. r1(X); r3(X); w3(X); w1(X); r2(X);*
*c. r3(X); r2(X); w3(X); r1(X); w1(X);*
*d. r3(X); r2(X); r1(X); w3(X); w1(X);*

4) Considere as três transações T1, T2 e T3 e os escalonamentos S1 e S2 descritos abaixo. Esboce os grafos de seriação (precedência) para S1 e S2. Verifique se são seriáveis e se for o caso determine os respectivos escalonamentos seriais.
T1: r1(X); r1(Z); w1(X);
T2: r2(Z); r2(Y); w2(Z); w2(Y);
T3: r3(X); r3(Y); w3(Y);
S1: r1(X); r2(Z); r1(Z); r3(X); r3(Y); w1(X); w3(Y); r2(Y); w2(Z); w2(Y);
S2: r1(X); r2(Z); r3(X); r1(Z); r2(Y); r3(Y); w1(X); w2(Z); w3(Y); w2(Y);

5) Considere as duas transações abaixo com valores iniciais de X=Y=0. É possível encontrar um escalonamento seriável equivalente ao escalonamento serial T0, T1 ?

```
T0: read(X);
    read(Y);
    if X = 0 then (Y = Y+1, write(Y));

T1: read(Y);
    read(X);
    if Y = 0 then (X = X+1; write(X));
```

6) Mostre que o protocolo de bloqueio em duas fases garante a seriação de conflitos de escalonamentos. Também mostre que tal protocolo também garante escalonamentos estritos (escalonamentos nos quais as transações não podem ler nem gravar um item X até que a última transação que gravou X tenha sido confirmada ou abortada).

# Referências Bibliográficas

[1] E. F. Codd. *A Relational Model of Data for Large Shared Data Banks*. Revista *CACM* volume = 6,1970.

[2] C. J. Date. Introdução a Sistema de Banco de Dados. Editora Campus, 2000

[3] S. Sudarshan and A. Silberschatz and F. Henry Korth. Sistemas de Banco de Dados. 3ª. Edição. Editora Makron Books, 1999

[4] E. R. Elmasri and S. Navathe and. *Fundamentals of Database Systems*. Editora Addison Wesley Pub, 2001.

[5] Jennifer Widow and Jeffrey Ullman *.A First Course in Database Systems*. Editora Prentice Hall, 1997.

[6] M Jackson *.Thirty years (and more) of databases* .Revista *Information and Software Technology*, volume = "41", páginas =969-978, 1999.

[7] R. Ramakrishnan and J. Gehrke. Database Management Systems. 2a. Edição. McGraw-Hill, 2000.

[8] C. Adamson, M. Venerable. *Data Warehouse Design Solutions*, John Wiley & Sons, 1998.

[9] E. Baralis, S. Paraboschi, E. Teniente. "Materialized view selection in a Multidimensional Database". *Proceedings of the 23rd VLDB Conference*, Atenas, Grécia, 1997.

[10] A. Gupta e I. S. Mumick. "What is the data warehousing problem? (Are materialized views the answer?)". *Proceedings of the 22nd VLDB Conference*, Mumbai (Bombay), Índia, 1996.

[11] M. Golfarelli, D. Maio, S. Rizzi. "Conceptual Design of Data Warehouses from E/R Schemes", *Proceedings of the Hawaii International Conference on System Sciences*, Kona, Hawaii, USA, 1998.

[12] *Information Advantage, Decision Path™Implementation Methodology.* "MyEureka!™ data warehouse requirements guide", 1999.

[13] W. H. Inmon, R. D. Hackathorn. *Como usar o Data Warehouse*. Infobook, Rio de Janeiro, 1997.

[14] S. Kelly. *Data Warehousing The Route to Mass Customization*, John Wiley & Sons, 1994.

[15] R. Kimball. *The data warehouse toolkit.* John Wiley & Sons, 1996.

[16] N. Raden. "Technology Tutorial: Modeling a Data Warehouse", disponível em: http://techweb.cmp.com/iw/564/64oldat.htm, 1996.

[17] S. Samtani, M. Mohania, V. Kumar, Y. Kambayashi. *ER Workshops*, 1998, pág. 81-92.

[18] R. Tanler. *The Intranet Data Warehouse*, John Wiley & Sons, 1997.

[19] Y. Zhuge, H. Garcia-Molina, J. Hammer e J. Widom*. View maintenance in a warehousing environment*. Technical report, Stanford University. Disponível: ftp://db.stanford.edu como pub/zhuge/1994/anomaly-full.ps. Outubro de 1994.